May 2019

Analyzing And Assuring Missions and Systems by STORM: Introducing and analyzing Systems-Theoretic and Technical Operational Risk Management (STORM)

Lori Denise Pickering
Syracuse University

Follow this and additional works at: https://surface.syr.edu/thesis
Part of the Engineering Commons

Recommended Citation
https://surface.syr.edu/thesis/317

This is brought to you for free and open access by SURFACE. It has been accepted for inclusion in Theses - ALL by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.
Abstract

The complexity of today’s large, multi-component systems and missions presents a growing risk of failure because of emergent system-level properties. Furthermore, the interconnectivity of systems to other systems creates additional security problems. Yesterday’s safety and security risk analysis methodologies are no longer effective. To manage this complexity, what is needed is a holistic, thorough, systematic, system-level, and formally verified approach to risk analysis to ensure stakeholder-required needs are met, asset losses are mitigated, and the system or mission operates with its intended functionality. Furthermore, these system and mission risks need to be thoroughly documented to increase the visibility of risks so that decision makers have a solid foundation upon which to base risk-mitigating decisions. Finally, the results of the analysis and decisions need to be formally verified and documented for the purpose of auditing and accountability.

This thesis presents a solution to this problem, System-theoretic and Technical Operational Risk Management (STORM). STORM is a methodology for designing trustworthy systems and missions that conform to industry standards of trustworthiness, namely the NIST SP 800-160 System Security Engineering Framework. It is also comformable to the Risk Management Framework (NIST SP 800-37).

Components of STORM have been successfully demonstrated on automated systems. But testing STORM on a non-automated, human-centered system has yet to be done. This paper demonstrates STORM analysis on the U.S. Army Ranger patrol base operations, an example of such a system. Following the example, this thesis discusses STORM in light of conformance to NIST SP 800-160. It also discusses improvements to STORM that could extend it to a more comprehensive system and mission assurance methodology. This could be done by explicitly adding components of the risk management framework (RMF NIST SP 37 and 800-53) and upgrading its documentation requirements based on the Assurance Case (AC) Methodology [1]. These additions would strengthen STORM’s trustworthiness component.
Analyzing And Assuring Missions and Systems

by STORM:

Introducing and analyzing

Systems-Theoretic and Technical Operational Risk Management
(STORM)

by

Lori Denise Pickering

B.S. University of California, Irvine, 2006
M.S., Syracuse University, 2017

Thesis
Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Syracuse University
May 2019
Acknowledgements

This research began in the summer of 2017 as part of the Assured by Design (ABD) program funded by the United States Air Force Research Laboratory (AFRL) in Rome, NY and managed by the principal investigator Professor Shiu-Kai Chin from the College of Engineering and Computer Science at Syracuse University. This project was envisioned by Professor Shiu-Kai Chin to satisfy the needs of the ABD program. This thesis evolved directly from this work.

Thanks and recognition go to the following people for their contribution to this project. Professor Shiu-Kai Chin for providing me with the opportunity and for his faith in me and my capabilities on this project. Erich Devendorf at AFRL for making the ABD program happen. Mizra Tihic for making this happen, especially with respect to funding.

A significant contributor to this research is U.S. Army Captain Jesse Nathaniel Hall who is also a graduate student at Syracuse University in the School of Information Science (iSchool). His translation of the patrol base operations from the U.S. Ranger Handbook is a significant contribution and it is noted where appropriate in this thesis.

Another contributor to this research is YiHong Guo, an undergraduate student at Syracuse University in the College of Engineering and Computer Science. YiHong’s contribution includes the original documentation\(^1\) of this work in LaTeX.

\(^1\)YiHong’s work does not appear in this thesis. However, he was very helpful and deserves some credit.
Disclaimer

The views in this thesis are that of the author’s. They do not in anyway represent the views of the United States Air Force Research Laboratory (AFRL) or the College of Engineering and Computer Science at Syracuse University.
# Table Of Contents

Abstract

<table>
<thead>
<tr>
<th>Table Of Contents</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xii</td>
</tr>
<tr>
<td>List of Acronyms</td>
<td>xiv</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 The Need for A Comprehensive Mission Assurance Methodology .......................... 1
1.2 Related Work ......................................................................................................... 3
1.3 Patrol Base Operations ......................................................................................... 4

### I System-theoretic Operational Risk Management (STORM) 8

### 2 STORM 9

### 3 System-Theoretic Process Analysis (STPA/STPA-Sec) 11

3.0.1 STAMP ................................................................................................................ 11
3.0.1.1 Core Components .............................................................................................. 13
3.0.2 STPA/STPA-Sec ...................................................................................................... 15
3.1 STPA/STPA-Sec Overview: Four Steps ....................................................................... 16

### 4 Certified Security by Design (CSBD) 19

4.1 Access Control Logic ............................................................................................... 19

### 5 Secure State Machine Model 31

### II STORM on Patrol Base Operations 41

### 6 STPA/STPA-Sec on Patrol Base Operations 42

6.1 Step 1: Define The Purpose of The Analysis .......................................................... 42
6.2 Step 2: Model The Control Structure ....................................................................... 49
6.3 Step 3: Identify Unsafe Control Actions (UCAs) .................................................... 55
6.3.1 Unsafe Control Actions (UCAs): Thomas Model .................................................. 57
6.4 Step 4: Identify Loss Scenarios .............................................................................. 60
6.4.1 Scenarios .......................................................... 60
6.5 Discussion And Conclusions ........................................... 72

7 Patrol Base Operations .................................................. 73
  7.1 Patrol Base Operations ................................................. 73
  7.2 Overview of The Model ............................................... 74
  7.3 Hierarchy of Secure State Machines .................................. 76
    7.3.1 Diagrammatic Description in Visio ................................ 76
    7.3.2 Descriptions of Individual Modules ................................. 78
    7.3.3 OMNI-Level ....................................................... 81
    7.3.4 Escape .......................................................... 82
    7.3.5 Top Level ....................................................... 83
    7.3.6 Horizontal Slice ................................................ 85
      7.3.6.1 ssmPlanPB .................................................. 85
      7.3.6.2 ssmMoveToORP .............................................. 87
      7.3.6.3 ssmConductORP .............................................. 88
      7.3.6.4 ssmMoveToPB ............................................... 89
      7.3.6.5 ssmConductPB ............................................... 90
    7.3.7 Vertical Slice .................................................. 91
      7.3.7.1 ssmSecureHalt ............................................... 92
      7.3.7.2 ssmORPRecon ............................................... 93
      7.3.7.3 ssmMoveToORP4L ............................................ 94
      7.3.7.4 ssmFormRT ............................................... 95
  7.4 Discussion .......................................................... 95

8 Patrol Base Operations as Secure State Machines .................... 97
  8.1 ssmPB: A Typical Example from the Hierarchy ....................... 98
    8.1.1 Principals ..................................................... 99
    8.1.2 States ........................................................ 99
    8.1.3 Outputs ....................................................... 100
    8.1.4 Commands ..................................................... 101
    8.1.5 Next-State Function .......................................... 102
    8.1.6 Next-Output Function ......................................... 103
    8.1.7 Authentication ................................................ 103
    8.1.8 Authorization ................................................ 106
    8.1.9 Proved Theorems ............................................... 109
  8.2 ssmConductORP: Multiple Principals .................................. 121
    8.2.1 Principals ..................................................... 122
    8.2.2 States ........................................................ 122
    8.2.3 Outputs ....................................................... 123
    8.2.4 Commands ..................................................... 123
    8.2.5 Next-State Function .......................................... 124
    8.2.6 Next-Output Function ......................................... 125
    8.2.7 Authentication ................................................ 125
    8.2.8 Authorization ................................................ 127
    8.2.9 Proved Theorems ............................................... 129
  8.3 ssmPlanPB: Non-sequential Transitions ............................... 140
    8.3.1 Principals ..................................................... 141
8.3.2 States ........................................ 142
8.3.3 Output ....................................... 142
8.3.4 Commands .................................... 143
8.3.5 Next-State Function ......................... 144
8.3.6 Next-Output Function ....................... 146
8.3.7 Authentication ............................... 147
8.3.8 Authorization ................................ 147
8.3.9 Proved Theorems ............................. 149
8.4 Discussion .................................... 162

III Discussion, Conclusion, and Future Work .......................... 163

9 STORM Conformance to SSE Framework ......................... 164
  9.1 Problem ........................................ 165
  9.2 Solution ....................................... 167
  9.3 Trustworthiness ................................ 168

10 Extending STORM to A Comprehensive Assurance Methodology 170
  10.1 STORM on The Risk Management Framework ............... 170
    10.1.1 Risk Management Framework (NIST SP 800-37) ........ 171
      10.1.1.1 Risk Categories ..................... 172
      10.1.1.2 Control Selection ................... 173
      10.1.1.3 Implementation ...................... 174
      10.1.1.4 Assess ................................ 174
      10.1.1.5 Authorize ............................ 174
      10.1.1.6 Monitor .............................. 174
  10.2 STORM Documentation Upgrade: The Assurance Case Methodology 175
    10.2.1 STORM Compared to The AC Methodology ............ 176

11 Future Work ..................................... 180
  11.1 Patrol Base Operations .......................... 180
  11.2 CSBD on Patrol Base Operations ................ 181
    11.2.0.1 Authentication ........................ 181
    11.2.0.2 Roles .................................. 181
  11.3 STORM ....................................... 181
    11.3.1 Automation ................................ 182

Appendices .......................................... 184

A Access Control Logic Theories: Pretty-Printed Theories .......... 185

B Parametrizable Secure State Machine & Patrol Base Operations: Pretty-Printed Theories 208

C Parametrizable Secure State Machine Theories: HOL Script Files 323
  C.1 ssm ........................................... 323
  C.2 satList ....................................... 328
## D Secure State Machine Theories Applied to Patrol Base Operations:

### HOL Script Files

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1</td>
<td>OMNILevel</td>
</tr>
<tr>
<td>D.2</td>
<td>TopLevel</td>
</tr>
<tr>
<td>D.2.1</td>
<td>PBTypeIntegrated Theory: Type Definitions</td>
</tr>
<tr>
<td>D.2.2</td>
<td>PBIntegratedDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.2.3</td>
<td>ssmPlanPBIntegrated Theory: Theorems</td>
</tr>
<tr>
<td>D.3</td>
<td>Horizontal Slice</td>
</tr>
<tr>
<td>D.3.1</td>
<td>ssmPlanPB</td>
</tr>
<tr>
<td>D.3.1.1</td>
<td>PlanPBTType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.3.1.2</td>
<td>PlanPBDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.3.1.3</td>
<td>ssmPlanPB Theory: Theorems</td>
</tr>
<tr>
<td>D.3.2</td>
<td>ssmMoveToORP</td>
</tr>
<tr>
<td>D.3.2.1</td>
<td>MoveToORPType Theory: Type Definitions &amp; Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.3.2.2</td>
<td>ssmMoveToORP Theory: Theorems</td>
</tr>
<tr>
<td>D.3.3</td>
<td>ssmConductORP</td>
</tr>
<tr>
<td>D.3.3.1</td>
<td>ConductORPType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.3.3.2</td>
<td>ConductORPDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.3.3.3</td>
<td>ssmConductORP Theory: Theorems</td>
</tr>
<tr>
<td>D.3.4</td>
<td>ssmMoveToPB</td>
</tr>
<tr>
<td>D.3.4.1</td>
<td>MoveToPBTType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.3.4.2</td>
<td>MoveToPBDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.3.4.3</td>
<td>ssmMoveToPB Theory: Theorems</td>
</tr>
<tr>
<td>D.3.5</td>
<td>ssmConductPB</td>
</tr>
<tr>
<td>D.3.5.1</td>
<td>ConductPBTType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.3.5.2</td>
<td>ConductPBDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.3.5.3</td>
<td>ssmConductPB Theory: Theorems</td>
</tr>
<tr>
<td>D.4</td>
<td>Vertical Slice</td>
</tr>
<tr>
<td>D.4.1</td>
<td>ssmSecureHalt</td>
</tr>
<tr>
<td>D.4.1.1</td>
<td>SecureHaltType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.4.1.2</td>
<td>SecureHaltDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.4.1.3</td>
<td>ssmSecureHalt Theory: Theorems</td>
</tr>
<tr>
<td>D.4.2</td>
<td>ssmORPRecon</td>
</tr>
<tr>
<td>D.4.2.1</td>
<td>ORPReconType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.4.2.2</td>
<td>ORPReconDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
<tr>
<td>D.4.2.3</td>
<td>ssmORPRecon Theory: Theorems</td>
</tr>
<tr>
<td>D.4.3</td>
<td>ssmMoveToORP4L</td>
</tr>
<tr>
<td>D.4.3.1</td>
<td>MoveToORP4LType Theory: Type Definitions</td>
</tr>
<tr>
<td>D.4.3.2</td>
<td>MoveToORP4LDef Theory: Authentication &amp; Authorization Definitions</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Relationship between System-Theoretic Process Analysis for Security (STPA-Sec), System-Theoretic Process Analysis (STPA), and System-Theoretic Accident Model and Processes (STAMP). (Image from Dr. William Young and Reed Prada 2017 STAMP Conference presentation in Boston, MA on March 27, 2017 [2]) ................................................................. 12

3.2 Example of a control model. (Image captured from the STPA Handbook [3].) .............................................................................. 14

3.3 Controller with process model. (Image captured from the Engineering a Safer World Handbook [4].) ........................................... 15

3.4 Four step process of STPA. (Image captured from the STPA Handbook [3].) .............................................................. 16

3.5 Control structure with potential system-level vulnerabilities/hazards and security considerations. (Image captured from Systems thinking for safety and security [5].) .................................................. 17


4.6 The HOL implementation of a Kripke structure (Princ). Image from Certified Security by Design Using Higher Order Logic [7] ................. 28


4.8 A partial description of the HOL implementation of the Kripke semantics (evaluation function). Image from Certified Security by Design Using Higher Order Logic [7] ................................................. 30


5.1 State machine versus secure state machine with a monitor. Image taken from Certified Security by Design Using Higher Order Logic [7]. ...... 33


5.3 TR rule for the exec transition rule in HOL. ................................. 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>TR_exec_cmd_rule for the <code>exec</code> transition rule in HOL.</td>
</tr>
<tr>
<td>5.5</td>
<td>TR_discard_cmd_rule for the <code>discard</code> transition rule in HOL.</td>
</tr>
<tr>
<td>6.1</td>
<td>Control structure for patrol base operations.</td>
</tr>
<tr>
<td>7.1</td>
<td>A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel Hall.</td>
</tr>
<tr>
<td>7.2</td>
<td>Diagrammatic description of patrol base operations as a modularized hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)</td>
</tr>
<tr>
<td>7.3</td>
<td>Escape level diagram.</td>
</tr>
<tr>
<td>7.4</td>
<td>Top level diagram.</td>
</tr>
<tr>
<td>7.5</td>
<td>Horizontal slice: PlanPB diagram.</td>
</tr>
<tr>
<td>7.6</td>
<td>Horizontal slice: MoveToORP diagram.</td>
</tr>
<tr>
<td>7.7</td>
<td>Horizontal slice: ConductORP diagram.</td>
</tr>
<tr>
<td>7.8</td>
<td>Horizontal slice: MoveToPB diagram.</td>
</tr>
<tr>
<td>7.9</td>
<td>Horizontal slice: ConductPB diagram.</td>
</tr>
<tr>
<td>7.10</td>
<td>Vertical slice: SecureHalt diagram.</td>
</tr>
<tr>
<td>7.11</td>
<td>Vertical slice: ORPRecon diagram.</td>
</tr>
<tr>
<td>7.12</td>
<td>Vertical slice: MoveToORP4L diagram.</td>
</tr>
<tr>
<td>7.13</td>
<td>Vertical slice: FormRT diagram.</td>
</tr>
<tr>
<td>8.1</td>
<td>Top level diagram.</td>
</tr>
<tr>
<td>8.2</td>
<td>Horizontal slice: ConductORP diagram.</td>
</tr>
<tr>
<td>8.3</td>
<td>Horizontal slice: PlanPB diagram.</td>
</tr>
</tbody>
</table>
## List of Tables

6.1 Organization goals. .................................................. 43
6.3 System accidents/losses. ........................................... 47
6.4 System-level hazards/vulnerabilities and constraints. ............... 48
6.5 Scenarios for UCA E1A3. Diagram by Captain Jesse Nathaniel Hall. 49
6.6 Controllers and process model. .................................... 50
6.7 Unsafe control actions unsafe control actions (UCAs) for control action "PL says planComplete." ........................................... 57
6.8 Unsafe control actions UCAs for control action "exec(planComplete)." 58
6.9 Unsafe control actions UCAs for control action "discard(anyCommand)." 59
6.10 Scenarios for UCA P1B1. (GOTWA = "(G) where I’m Going, (O) Others I’m taking, (T) Time of my return, (W) What to do if I don’t return, (A) Actions to take if I’m hit or Actions to take if you’re hit (5 point contingency plan)" [10]). ........................................... 62
6.11 Scenarios for UCA P1B1/P1C1. (METT-TC = mission, enemy, terrain, troops available, time, and civilian considerations [11]). ............... 63
6.12 Scenarios for UCA P1A1. ........................................... 64
6.13 Scenarios for UCA P1A2. ........................................... 65
6.14 Scenarios for UCA P1A3. ........................................... 66
6.15 Scenarios for UCA P1A4. ........................................... 66
6.16 Scenarios for UCAs E1B1 and E1C1. ................................ 67
6.17 Scenarios for UCA E1A1. ........................................... 68
6.18 Scenarios for UCA E1A2. ........................................... 69
6.19 Scenarios for UCA E1A3. ........................................... 70
6.20 Scenarios for UCA E1A4. ........................................... 70
6.21 Scenarios for UCA for discard on all commands (D1A1 through D1A4). ........................................... 71

### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>access-control logic.</td>
</tr>
<tr>
<td>CAST</td>
<td>Causal Analysis based on Systems Theory.</td>
</tr>
<tr>
<td>CSBD</td>
<td>Certified Security by Design.</td>
</tr>
<tr>
<td>HOL</td>
<td>Higher Order Logic Theorem Prover.</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology.</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology.</td>
</tr>
<tr>
<td>ORP</td>
<td>Objective Rally Point.</td>
</tr>
<tr>
<td>SSE</td>
<td>Systems Security Engineering.</td>
</tr>
<tr>
<td>SSM</td>
<td>secure state machine.</td>
</tr>
<tr>
<td>STAMP</td>
<td>System-Theoretic Accident Model and Processes.</td>
</tr>
<tr>
<td>STORM</td>
<td>System-Theoretic and Technical Operational Risk Management.</td>
</tr>
<tr>
<td>STPA</td>
<td>System-Theoretic Process Analysis.</td>
</tr>
<tr>
<td>UCA</td>
<td>unsafe control actions.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 The Need for A Comprehensive Mission Assurance Methodology

Without adequate assurance methodologies, costly or embarrassing system flaws can go to market such as the Meltdown [12] and Spectre [13] flaws. These flaws had the potential to allow unauthorized access of private information such as passwords. For military missions, missions could fail, resources could be squandered, lives could be lost. System and mission assurance is critical.

Building assured systems presents major problems in today’s systems because today’s systems are growing in both size and complexity. Furthermore, everything is connected to everything else. Consider the following problem excerpt from the STPA Handbook [3].

Some Navy aircraft were ferrying missiles from one point to another. One pilot executed a planned test by aiming at the aircraft in front (as he had
been told to do) and firing a dummy missile. Apparently nobody knew that
the “smart” software was designed to substitute a different missile if the one
that was commanded to be fired was not in a good position. In this case,
there was an antenna between the dummy missile and the target, so the
software decided to fire a live missile located in a different (better) position
instead. What aircraft component(s) failed here?

All components performed their function correctly, but the system as a whole (including
the pilot and the pilot’s training) failed. What this emphasizes is the need for a an
assurance methodology that tackles the problem at a system-level. The system-theoretic
approach to assuring systems is tackled by STPA/STPA-Sec, a component of STORM.

Furthermore, this assurance methodology should bring together the needs of the policy
makers, who define the mission and system functionality and requirements, and the
technical crew who design and develop the system from the mission’s concepts of
operations (CONOPS). This is also what STPA/STPA-Sec does.

Once developed, these CONOPS should be verified to be trustworthy using formal
theorem proving methods so that the trustworthiness of the CONOPS can be verified
and documented for accountability. CSBD, the second component of STORM, does this.

Finally, a comprehensive mission assurance methodology is also needed to meet the
multiple Department of Defense (DoD) component needs as stated in the Mission
Assurance (MA) Strategy by the U.S. Deputy Secretary of Defense in 2012 [14].
STORM should be a part of this strategy.

As the complexities of systems and organizations grow, a systems-level and overall
comprehensive strategy is necessary to help decision makers meet their mission
assurance goals.
This thesis introduces STORM and applies the STORM methodology to an example of a non-automated, human-centered system. This system is the U.S. Army patrol base operations as described in the U.S. Army Ranger Handbook. Until this thesis, STORM had not been applied to this type of mission. After demonstrating STORM on the patrol base operations, this thesis demonstrates how STORM conforms to the SSE Framework. Afterwards, it discusses how components of the Assurance Case (AC) Methodology could improve STORM’s trustworthiness component. Afterwards, this thesis suggests explicitly adding components of the Risk Management Framework (RMF) to STORM. Finally, this thesis suggests how STORM could be developed into a more comprehensive mission assurance strategy that could satisfy the needs of the DoD’s Mission Assurance Strategy Framework as outline in [14].

1.2 Related Work

There are other efforts to develop system and mission assurance strategies. This thesis does not present a comprehensive review of the literature in this area. However, one such methodology showed promise. Kawakami et. al. did a thorough search of literature and found a need for an assurance methodology in computer architecture. They presented their solution to the problem called the Assurance Case Methodology. This methodology describes security goals, breaks them down into subgoals, presents a rationale for meeting the security goal, and then finally provides evidence that the security goal was met. While the AC Methodology is not a holistic and thorough methodology, aspects of the methodology suggest areas of improvement for STORM, specifically with respect to documentation. A comparison of STORM and the AC Methodology is discussed in Chapter 10. Lessons learned from the AC Methodology are suggested as upgrades to STORM’s documentation.

More generally, numerous works by NIST define standards and frameworks for various
fields of science and technology fields. Those most relevant to this thesis are the NIST SP 800-160 and 800-37. These are discussed with respect to STORM in Part III of this thesis.

1.3 Patrol Base Operations

Analyzing system and mission security properties with components of STORM has been successfully demonstrated prior to this work. For example, CSBD has been applied to automated credential verification in financial systems for JP Morgan [15] and to verify the command and control systems in the U.S. Air Force F16 Viper.\(^1\) STPA-Sec is currently U.S. Air Force doctrine. Nonetheless, demonstrating STORM on a purely non-automated and human-centered system has been uncharted territory. This thesis demonstrates STORM on such a system, analyzing the mission’s safety and security features and proving properties of complete mediation (i.e., actions are taken if and only if they are authenticated and authorized). The analysis is a top-level analysis of an abstracted model of the patrol base operations intended to assess the efficacy of STORM on such a system.

The theorems proved in the CSBD portion of this paper demonstrate the logical soundness of the patrol base operations concepts of operation CONOPS. In particular, they prove that actions are taken if and only if those actions are both issued by an authenticated person and authorized by the mission’s policy.

For example, if the Platoon Leader has completed the planning phase of the mission and is ready to cross the line of discrimination, the Platoon Leader issues the command

\(^1\)Quote from professor Shiu-Kai Chin: "AFRL: UAV payload controller and secure memory loader verifier (SMLV) for F-16 Viper" not published, tech papers only.
Satisfying the access-control policy means that the Platoon Leader must be authenticated by some means designated in the access-control policy. The access-control policy must also state which commands the Platoon Leader is authorized to issue. In the access-control logic (ACL) discussed in chapter 4, this is represented as

\[ \text{PlatoonLeader controls crossLD} \]

Satisfying both of these conditions combined with the Platoon Leader’s request to cross the line of discrimination results in the following logical sequence and conclusion.

\[ \text{if PlatoonLeader says crossLD and} \]
\[ \text{PlatoonLeader controls crossLD, then} \]
\[ \text{crossLD is justified} \]

The CSBD uses formal theorem proving methods to demonstrate and verify this property. Furthermore, it also demonstrates and verifies the converse.

\[ \text{if crossLD is justified} \]
\[ \text{then PlatoonLeader says crossLD and} \]
\[ \text{PlatoonLeader controls crossLD} \]

This theorem demonstrates the property of complete mediation, namely that access and modification of all security-sensitive objects is monitored and controlled. Verifying complete mediation is key because it demonstrates that the system is trustworthy with
respect to access-control. Documenting complete mediation is also the key to accountability. The CSBD provides verifiable and reproducible proof of complete mediation to serve this purpose.

STPA/STPA-Sec demonstrates a thorough analysis of the patrol base operations safety and security vulnerabilities. It identifies the vulnerabilities and unsafe and insecure actions that the patrol base operations may encounter. It also suggests measures to mitigate or avoid these unsafe or insecure actions.

For example, if the Platoon Leader does not complete the planning in a timely manner, the patrol base operations may conduct the wrong mission (go after the wrong objective) or the mission may fail. In addition, if the proper equipment, back-up support, and contingency planning are not conducted at the right time then the Platoon mission may fail and lives could be lost.

To mitigate these unacceptable events, the analysis considers possible scenarios that could cause them. For example, the Platoon Leader may have too many obligations. One solution to this is to limit the Platoon Leader’s span of control and delegate work to subordinates.

Ineffective communication could also cause the planning phase to be incomplete or not conducted in a timely manner. This could be caused by faulty equipment or a lack of communication regarding communications protocols. To mitigate this, check-lists could be used to verify which equipment is needed for the various types of missions and how that equipment should be tested prior to mission. In addition, communications protocols such as key words and communications channels could be established before commencement of the mission.

Discovering these hazards and vulnerabilities requires a thorough and systematic methodology to analyze what could go wrong in the mission. This is what the
STAP/STPA-Sec prescribes, using a methodology that is founded on the philosophy that hazards and vulnerabilities are caused by unsafe or insecure control actions.

This thesis tests the STORM methodology on a system that it had yet been applied to. It applies the methodology on an example of a non-automated, human-centered system. It also makes suggestions on how additional components could be added to STORM to bring it further into conformance with the Risk Management Framework (NIST Special Publication 800-37). It also suggests adding evidence (AC Methodology) to step 4 of the STPA/STPA-Sec analysis to strengthen the trustworthiness component of conformance to the System Security Engineering Framework (NIST SP 800-160).

This thesis begins with an introduction to STORM and its components in part I. Following this, STORM is applied to the patrol base operations in part II. Finally, in part III, STORM is discussed in the context of conformance to the SSE Framework. Improvements to STORM based on the RMF and AC Methodology are also discussed. Part III concludes with suggestions for future work.
Part I

System-theoretic Operational Risk Management (STORM)
Chapter 2

STORM

This chapter provides some background on System-Theoretic and Technical Operational Risk Management (STORM) and its components. STORM aligns itself with the SSE Framework which is demonstrated explicitly and discussed in more detail in chapter 10.

STORM is a system and mission assurance methodology that applies hazard and security analysis tools and formal theorem proving methods to design and verify trustworthy systems. It conforms to the System Security Engineering Framework (see Chapter 10). STORM assists the analyst in developing concepts of operations (CONOPS) that are secure and trustworthy.

STORM begins with an analysis of organizational and stakeholder needs and their definition of assets and asset losses. It identifies unacceptable asset losses. It defines the system and then describes the system’s functional processes and process models. From here, it identifies system or mission hazards and vulnerabilities and then develops constraints. The process is iterated and further refined. Causal models of hazards and vulnerabilities are investigated and rationale are identified to mitigate them. This analysis is then developed into a CONOPS. Access-control policies are developed for the CONOPS which are then formally verified and documented.
STORM is the result of the collaborative effort of Professor Shiu-Kai Chin of the College of Engineering and Computer Science at Syracuse University, Erich Devendorf, PhD of the U.S. Air Force Research Laboratory, and Col. William Young, PhD of the U.S. Air Force 53rd Electronic Warfare Group.

STORM is composed of two components which implement the steps described above. The first is System-theoretic process analysis modified for security (STPA-Sec). The second is certified security by design (CSBD). System-Theoretic Process Analysis (STPA) is the culmination of years of research in safety engineering by Massachusetts Institute of Technology (MIT) professor Nancy Leveson and MIT researcher John Thomas, PhD. It is a response to the failure of legacy methods to manage the complexities that arise in today’s complex systems. System-Theoretic Process Analysis for Security (STPA-Sec) is a modified version of STPA that focuses on security. It is the work of Dr. Young, who presented it as his PhD thesis in 2014 at MIT. STPA-Sec is currently U.S. Air Force doctrine.

Certified Security by Design (CSBD) is derived from the work of Professor Shiu-kai Chin and Professor Susan Older (also of the College of Engineering and Computer Science at Syracuse University). CSBD has been successfully used in industry. It was used in the design of JP Morgan Company’s SWIFT protocols [15] and on F-16 Viper secure memory loader verifier (not published).

The next chapters describe the two components of STORM in more detail.
Chapter 3

System-Theoretic Process Analysis
(STPA/STPA-Sec)

The complexity of today’s multicomponent and multi-connected systems required a relook into accident and security analysis. This formed the basis for research that ultimately culminated in the System-theoretic process analysis (STPA). STPA-Sec is a modification to STPA for security analysis. The full details of STPA can be found in the STPA Handbook [3]. This section describes the philosophy underlying STPA (STAMP accident model) and the overall process.

3.0.1 STAMP

Figure 3.1 shows the relationship between STPA-Sec, STPA, and System-Theoretic Accident Model and Processes (STAMP).

STAMP is a way of thinking about how accidents occur. It assumes both a traditional and a system-theoretic view. In the traditional view, accidents are caused by unsafe chains of events. In the system theoretic view, accidents are also caused by dynamic
and complex interactions.

The underlying philosophy of STAMP is that the current practice of analyzing individual component reliability and chain-of-causality failures does not capture the full breadth of hazards associated with today’s complex systems. These systems require a system-theoretic approach to hazard analysis. System theory focuses on the system as a whole rather than merely a collection of subcomponents. The need for a system theory approach is epitomized in the notion that the whole is greater than the sum of its parts. From the complex interactions of individual components arise emergent properties. These properties can be thought of as a higher order that is not predictable from the behavior of the individual components.

STAMP forms the foundation of a different type of systems-theoretic safety analysis techniques. Several approaches to safety engineering are built upon the STAMP foundation. System-Theoretic Process Analysis (STPA) is one of these analysis techniques. In addition to STPA, several other analysis techniques are built on top of the STAMP foundation. These include Causal Analysis based on Systems Theory (CAST), STPA for security (STPA-Sec), STPA for privacy (STPA-Priv), STPA-SafeSec
[16], and SAFE (a refinement focused on hardware and software subsystems [16]).

STAMP is the subject of yearly conferences world wide promoted by the Partnership for Systems Approaches to Safety and Security (PSASS)\(^1\). When compared to other accident models, STAMP is more effective.\(^2\) It also cost less to implement [3].

### 3.0.1.1 Core Components

At STAMP’s core is a top-down model that views accidents as dynamic control problems. It focuses on safety constraints, hierarchical control structures, and process models [4].

**safety constraints**  STAMP views constraints, rather than events, as the safety-critical processes. A lack of constraints can lead to a hazardous system state which will lead to an accident in the worst-case scenario.

**hierarchical control structures**  Figure 3.2 is an example of a hierarchical control model. The flow of control is from top to bottom; higher-level control structures control lower-level control structures. Downward pointing arrows indicate instructions to lower level controllers or controlled processes. Upward pointing arrows indicate feedback from lower components.

At the top of the model are the high-level controllers. These controllers send commands to and receive feedback from controlled processes below. In the model in figure 3.2, the Human Controller(s) is at the top. This controller sends commands to both the Actuator(s) and the Automated Controller. The Human Controller(s) receives feedback

\(^1\)See [https://psas.scripts.mit.edu/home/other-stamp-meetings/](https://psas.scripts.mit.edu/home/other-stamp-meetings/).

\(^2\)See, for example, Paul Stukus’ thesis dissertation on how STAMP outdid other techniques when analyzing a U.S. Coast Guard Buoy Tender Integrated Control System [17]
from the Automated Controller and the Sensor(s). The Physical Process is a controlled process. It is controlled by the Actuator(s) and sends feedback via the Sensor(s).

Figure 3.2: Example of a control model. (Image captured from the STPA Handbook [3].)

The cause of accidents is inadequate control at higher levels, which trickle down to lower levels causing hazardous conditions.

**Inadequate Or Unsafe Controls** STAMP considers four types of inadequate or unsafe controls on constraints:

- insufficient or missing constraints
- wrong constraints
- out-of-order or under-timed constraints
- constraints that are imposed for too long.

These will translate into the four types of unsafe control actions in step 3 of the STPA analysis (described next).
**process models**  Figure 3.3 shows a controller with a process model. Each controller has its own process model. The process model describes the controller’s understanding of how the system works (i.e., internal algorithms) as well as knowledge about the state of the system (i.e., interpretation of feedback from other systems).

If the controller’s process model is somehow flawed or inconsistent with the system’s state, it could cause a hazardous condition which could cause unacceptable losses in the worst-case scenario.

![Controller with process model](Image captured from the Engineering a Safer World Handbook [4].)

**3.0.2 STPA/STPA-Sec**

STPA/STPA-Sec is a four step process that defines organizational and stakeholder needs, identifies assets and unacceptable asset losses, and identifies hazards or vulnerabilities. It constructs a functional control model and uses this to identify unsafe control actions and rationale for mitigating those actions. It is an iterative process that refines and updates itself based on new findings.
3.1 STPA/STPA-Sec Overview: Four Steps

Figure 3.4 diagrams these four steps [3].

Figure 3.4: Four step process of STPA. (Image captured from the STPA Handbook [3].)

Step 1: Define The Purpose of The Analysis  This step begins with a definition of the organizational and stakeholder needs. It then defines assets and unacceptable asset losses. These unacceptable losses identify the stakeholder’s definition of security. This critical step in the analysis ties the mission or system to its purpose. It defines what the "right mission" is. It links the stakeholders needs to the safety and security technical aspects of mission or system.

This step also defines the system and its inputs. In addition, system-level hazards and vulnerabilities are also identified. These can be refined throughout the process. System-level constraints are also defined. These constraints are also refined.

This step requires knowledge of the purpose of the system, the stakeholders involved, and the organizational needs.
Step 2: Model The Control Structure  This step develops the functional control model, control processes, controlled processes, and process models. Figure 3.5 shows a generic functional control model for a system or mission with an adversary (controller 2).

![Control Structure Diagram](Image)

Figure 3.5: Control structure with potential system-level vulnerabilities/hazards and security considerations. (Image captured from Systems thinking for safety and security [5].)

(1) and (3) type vulnerabilities/hazards are caused by representation flaws. These include missing or incorrect information or feedback and flaws involving the controller’s model of the system. (2) type vulnerabilities/hazards are caused by algorithm flaws (algorithms implement the controller’s process model). (4) type vulnerabilities/hazards are caused by component failures.

This step requires knowledge of the system or mission.
Step 3: Identify Unsafe Control Actions (UCAs)  The next step identifies unsafe control actions. These are the inadequate or unsafe controls described in the STAMP model. The four types of UCAs are

- Not applying the control action
- Applying the wrong control action
- Applying the control action in the wrong order
- Applying the control action for too long or not long enough.

Control actions are actions that a controller issues to another controller or controlled process. In the analysis, each possible action is analyzed for all four UCAs for each possible state of the controller and controlled process. This process is lengthy but thorough. It requires a system or mission expert.

Step 4: Identify Loss Scenarios  This last step identifies scenarios that could cause the UCAs in the previous step in the worse-case. It also provides rationale to avoid or mitigate the scenario.

This step requires a system or mission expert.

Chapter 6 demonstrates an STPA/STPA-Sec analysis on the patrol base operations. But before performing the analysis, the next chapter describes the second component of STORM, CSBD.
Chapter 4

Certified Security by Design (CSBD)

Certified Security by Design (CSBD) is a method for formally verifying and documenting the security properties of a system. It satisfies the principle of complete mediation. It uses an access-control logic (ACL) to reason about access to security sensitive objects. It uses computer-aided reasoning such as the Higher Order Logic (HOL) Interactive theorem prover to formally verify and document these security properties. The outcomes of CSBD are consistent with the Systems Security Engineering (Systems Security Engineering (SSE)) Framework described in National Institute of Standards and Technology (NIST) SP 800-160 [8]. They serve as a reproducible and auditable documentation of trustworthiness.

4.1 Access Control Logic

This section describes the access-control logic (ACL) for the CSBD. It begins with the basic concepts of the ACL. Then, it describes the ACL implementation in the Higher Order Logic (HOL) Interactive Theorem Prover. The explanation here is a simplified version of the ACL and describes only what is necessary to understand how the ACL is
The ACL is a formal propositional logic that specifically reasons about security-related objects. Security in this sense means confidentiality, integrity, and availability (CIA). The formal proofs generated by the ACL demonstrate that a system does (or does not) satisfy the security properties of the system. A full discussion of the ACL is the subject of the text *Access Control, Security, and Trust: A Logical Approach* by Syracuse University professors Shiu-Kai Chin and Susan Older. This section describes ACL in sufficient detail to gain a basic understanding of what the proofs mean and the logic from which they are derived.

At its most basic, the ACL is composed of principals that make requests to access objects. These principals are given authority or jurisdiction over objects. These principals must be authenticated before they can access or modify objects over which they have authority. The following sections describe the components and logical rules of the ACL.

**Principals, Requests, Authority, and Jurisdiction**

Principals should be thought of as actors in the access-control logic. Principals can make statements or requests. They can be assigned privileges or authority over objects or actions. More formally, principals are defined as

\[
\text{Princ ::= PName / Princ \& Princ / Princ} \mid \text{Princ}
\]

To reason about access-control and trust, the ACL uses propositional variables, requests, authority, and jurisdiction to make statements.

Principals can make requests. In the ACL principals make requests using the *says*
operator. Requests have the form $P \text{ says } \varphi$, where $P$ represents some principal and $\varphi$ represents some assertion (proposition, propositional variable). For example, 

*Commander says dropLeft*. In this example, the Commander is issuing a command (or request) for the munitions system to drop the left weapon.

Principals can have authority over assertions. The ACL conveys authority using the *controls* operator. Statements of authority have the form $P \text{ controls } \varphi$, where $P$ represents some principal and $\varphi$ represents some assertion. For example, *Commander controls dropLeft*. This states that the Commander has the authority over the command (or request) for the munitions system to drop the left weapon.

Principals can also have jurisdiction over assertions. Both authority and jurisdiction use the *controls* operator. Statements of jurisdiction have the same form as statements of authority. Statements of authority are typically defined in an organization’s policy. Statements of jurisdiction are statements that are readily believed given the context. For example, *PresidentOfUS controls (Commander controls dropLeft)*. In this example, the President of the United States, per the U.S. Constitution, has jurisdiction over the authority invested in the Commander. This is readily believed and does not require any further proof than "because the Constitution says so."

Additional concepts such as one principal speaking for another and one principal representing another on some assertion are also covered in the ACL. In addition, the concept of assigning integrity and security levels (such as the system used to assign U.S. security clearances to individuals) to principals and objects is covered by the ACL. However, these are not employed in our simple proof-of-concept version. They are further elaborated on in the text.
Syntax: Well-Formed Formulas

Well-formed formulas (WFFs) define the syntax (or grammatical structure) of the propositional logic. They are syntactically valid statements in the ACL. All ACL statements must be formulated as WFFs. The syntax is defined as follows.

\[
\text{Form ::= PropVar} / \neg \text{Form} / (\text{Form} \lor \text{Form}) / \\
(\text{Form} \land \text{Form}) / (\text{Form} \supset \text{Form}) / (\text{Form} \equiv \text{Form}) / \\
(\text{Princ} \Rightarrow \text{Princ}) / (\text{Princ says Form}) / (\text{Princ controls Form}) / \\
(\text{Princ reps Princ on Form})
\]

Semantics: Kripke Structures and Evaluation Function

Whereas WFFs define the syntax of the propositional logic, Kripke semantics with the evaluation function describes the semantics (or meaning).

A Kripke structure primarily deals with three things: worlds, propositions, and principals. The worlds can be thought of as possible states or configurations of some system. Propositions are just statements that are either true or false. And, principals are just actors. A Kripke structure \( \mathcal{M} = (W, I, J) \) is defined as a three-tuple consisting of the following: a set of worlds \( W \); a function \( I \) called the assignment function that maps propositions to worlds, and ; a function \( J \) that maps principals to relations on worlds, where the relation is called the accessibility relation. Formally, these are defined as follows

- \( W \) is a nonempty set, whose elements are called worlds.
- \( I : \text{PropVar} \rightarrow \mathcal{P}(W) \) is an interpretation function that maps each propositional variable to a set of worlds.

\(^1\)The last line is from [7].
\[
\begin{align*}
\mathcal{E}_M[p] &= I(p) \\
\mathcal{E}_M[\neg \varphi] &= W - \mathcal{E}_M[\varphi] \\
\mathcal{E}_M[\varphi_1 \land \varphi_2] &= \mathcal{E}_M[\varphi_1] \cap \mathcal{E}_M[\varphi_2] \\
\mathcal{E}_M[\varphi_1 \lor \varphi_2] &= \mathcal{E}_M[\varphi_1] \cup \mathcal{E}_M[\varphi_2] \\
\mathcal{E}_M[\varphi_1 \supset \varphi_2] &= (W - \mathcal{E}_M[\varphi_1]) \cup \mathcal{E}_M[\varphi_2] \\
\mathcal{E}_M[\varphi_1 \equiv \varphi_2] &= \mathcal{E}_M[\varphi_1 \supset \varphi_2] \cap \mathcal{E}_M[\varphi_2 \supset \varphi_1] \\
\mathcal{E}_M[P \Rightarrow Q] &= \begin{cases} W, & \text{if } \hat{J}(Q) \subseteq \hat{J}(P) \\
\emptyset, & \text{otherwise} \end{cases} \\
\mathcal{E}_M[P \text{ says } \varphi] &= \{ w | \hat{J}(P)(w) \subseteq \mathcal{E}_M[\varphi] \} \\
\mathcal{E}_M[P \text{ controls } \varphi] &= \mathcal{E}_M[(P \text{ says } \varphi) \supset \varphi] \\
\mathcal{E}_M[P \text{ reps } Q \text{ on } \varphi] &= \mathcal{E}_M[(P | Q \text{ says } \varphi) \supset Q \text{ says } \varphi]
\end{align*}
\]

Figure 4.1: Kripke semantics. Image taken from Access Control, Security, and Trust: A Logical Approach[6]

- \( J : \text{PName} \rightarrow \mathcal{P}(W \times W) \) is a function that maps each principal name to a relation on worlds.

The Kripke semantics define the meanings of WFFs for Kripke structures. The semantics can be thought of as an evaluation function for a particular Kripke \( \mathcal{M} = \langle W, I, J \rangle \). Figure 4.1 shows the Kripke semantics. The subscript \( \mathcal{M} \) signifies that the evaluation function is defined for a particular Kripke structure. This means there is a separate evaluation function for each Kripke structure.

**Satisfies and Soundness Properties**

The "satisfies" condition applies to a particular Kripke structure \( \mathcal{M} = \langle W, I, J \rangle \). It is said that \( \mathcal{M} \) satisfies some proposition \( \varphi \) if the evaluation function \( \varepsilon_{\mathcal{M}}[\varphi] = W \) (the set of all worlds) for \( \mathcal{M} \). In other words, \( \varphi \) is true in all worlds of \( \mathcal{M} \). Symbolically, this is denoted as \( \mathcal{M} \models \varphi \). The statement \( \mathcal{M} \) does not satisfy \( \varphi \) is denoted as \( \mathcal{M} \not\models \varphi \).
Whereas "satisfies" describes a property of a Kripke structure $\mathcal{M}$, "soundness" describes a property of all Kripke structures.

Soundness refers to the logical consistency of inference rules. Inference rules consist of a set of hypothesis $\{H_1, H_2, \ldots, H_n\}$ and a conclusion.

$$\frac{H_1, H_2, \ldots, H_n}{C}$$

An inference rule is said to be sound if for every Kripke structure $\mathcal{M}$ that satisfies all the hypotheses, the conclusion is true. In other words, the inference rule is sound if and only if $\forall H_i, \mathcal{M} \models H_i \rightarrow \mathcal{M} \models C$.

Soundness is verified by formal proofs that employ axioms, tautologies, and sound inference rules that are already proved.

**Inference Rules**

The inference rules for the ACL are shown in figure 4.2. All the inference rules are sound. Details of proofs of soundness can be found in *Access Control, Security, and Trust: A Logical Approach*[6].

The primary rule applied in the formal proofs for this project is the *Controls* rule, shown in figure 4.2 and shown again here in figure 4.3.

This rule has two hypotheses and one conclusion. The left hypothesis is an authorization.\(^2\) The principal $P$ controls (is authorized on) some assertion $\varphi$. The right hypothesis is a request.\(^3\) The principal $P$ requests some assertion $\varphi$. The conjunction of the authorization and the request of $P$ on $\varphi$ results in the assertion $\varphi$. That is, if $P$

\(^2\)or a *control* in the C2 calculus
\(^3\)or a *command* in the C2 calculus
Figure 4.2: The ACL inference rules. Image taken from Access Control, Security, and Trust: A Logical Approach[6]

controls $\varphi$ and $P$ says $\varphi$ then $\varphi$ is true. For example, if Commander controls dropLeft and Commander says dropLeft then dropLeft is justified.

Higher Order Logic (HOL)

This section describes how the access-control logic (ACL) is implemented in the Higher Order Logic (HOL) Interactive Theorem Prover.

Figure 4.4 shows the HOL representations for principals ($\text{Princ}$).

The definitions here correspond to the $\text{Princ}$ defined above. The first line corresponds to $\text{PropVar}$, the second corresponds to $\text{Princ} \& \text{Princ}$, and the third corresponds to $\text{Princ} \mid \text{Princ}$. In HOL, the infix $\&$ operator is represented with the prefix meet.
operator. The infix \( | \) operator is represented with the prefix quoting operator.

In the definition for \( Princ \), Name is called the type constructor. The result of the constructor and a concrete type or type variable results in something of type \( Princ \). Examples of principals in HOL are:

\[
\text{Name Commander, or } \quad (\text{Name Commander}) \; \text{ meet } \; (\text{Name MunitionsAvailable}), \text{ or } \\
(\text{Name Commander}) \; \text{ quoting } \; (\text{Name MunitionsAvailable}).
\]

The top declaration is the only one used in this project.

The HOL representation of WFFs is shown in figure 4.5.

\textbf{Form} is the datatype definition. \( TT \) and \( FF \) are the ACL representations of true and false, respectively. \textit{notf}, \textit{andf}, \textit{orf}, \textit{impf}, \textit{eqf}, \textit{says}, \textit{speaks\_for}, \textit{controls}, and \textit{reps} are all the prefix versions of the infix operators defined above. The additional elements of the \textit{Form} (from domi to the end) refer to integrity and security levels, which are not used for this project.

The type definitions that follow the operator are relevant and show-up everywhere in the code. It is useful to dissect one of them. Consider the \textit{andf} operator.

\[
(\text{andf}) \ (('\text{aavar}, '\text{apn}, 'il, 'sl) \text{ Form})
\]
Figure 4.5: The definition for \texttt{Form} in HOL. \textit{Certified Security by Design Using Higher Order Logic}\cite{7}

\begin{verbatim}
Form =
    | TT
    | FF
    | prop 'aavar
    | notf (('aavar, 'apn, 'il, 'sl) Form)
    | (andf) (('aavar, 'apn, 'il, 'sl) Form)
      (('aavar, 'apn, 'il, 'sl) Form)
    | (orf) (('aavar, 'apn, 'il, 'sl) Form)
      (('aavar, 'apn, 'il, 'sl) Form)
    | (imf) (('aavar, 'apn, 'il, 'sl) Form)
      (('aavar, 'apn, 'il, 'sl) Form)
    | (eqf) (('aavar, 'apn, 'il, 'sl) Form)
      (('aavar, 'apn, 'il, 'sl) Form)
    | (saf) ('apn Princ) (('aavar, 'apn, 'il, 'sl) Form)
    | (speaks_for) ('apn Princ) ('apn Princ)
    | (controls) ('apn Princ) (('aavar, 'apn, 'il, 'sl) Form)
    | reps ('apn Princ) ('apn Princ)
      (('aavar, 'apn, 'il, 'sl) Form)
    | (domi) ('apn, 'il) IntLevel (('apn, 'il) IntLevel)
    | (eqi) ('apn, 'il) IntLevel (('apn, 'il) IntLevel)
    | (doms) ('apn, 'sl) SecLevel (('apn, 'sl) SecLevel)
    | (eqs) ('apn, 'sl) SecLevel (('apn, 'sl) SecLevel)
    | (eqn) num num
    | (lte) num num
    | (lt) num num
\end{verbatim}

\((('aavar, 'apn, 'il, 'sl) Form)\) is the type signature for another \texttt{Form}. The component types of a \texttt{Form} are a proposition ('aavar), a principal ('apn), an integrity level ('il), and a security level ('sl). The prefix operator \texttt{andf} then takes two \texttt{Forms} each of type \((('aavar, 'apn, 'il, 'sl) Form)\). For the example used in this project, type \((('aavar, 'apn, 'il, 'sl) Form)\) looks like type \(((\text{command option}, \text{principal}, 'd, 'e) \text{Form})\).

The Kripke structure in HOL is shown in figure 4.6.

In figure 4.6, the first set of parenthesis represents the assignment function. It takes a
**Kripke** =

\[ \text{KS ('aavar -> 'aaworld -> bool) ('apn -> 'aaworld -> 'aaworld -> bool) ('apn -> 'il) ('apn -> 'sl)} \]

Figure 4.6: The HOL implementation of a Kripke structure (Princ). Image from *Certified Security by Design Using Higher Order Logic* [7]

proposition and world as arguments. If the proposition is true in that world, then the function returns true, otherwise it returns false. The second pair of parenthesis is similar to the accessibility function. It takes a principal, a world, and another world. The function returns true if the second world is accessible from the first for this particular principal, otherwise it returns false. The last two pairs of parentheses represent integrity and security levels. These are not used in this thesis.

In the HOL code provided for this project, Kripke structures are either abbreviated as follows

\[(M, Oi, Os)\]

Or, unabbreviated as follows

\[M: ('prop, 'world,'pName, 'Int, 'Sec)Kripke, (Oi: 'Int po), (Os:'Sec po)\]

The actual Kripke structure is represented by \(M\) and the integrity and security levels are represented by \(Oi\) and \(Os\).

The ACL formulas in HOL are shown in figure 4.7

Triangular brackets enclose propositions. Thus, a proposition codes as follows:

\[\text{prop command}\]

28
The other formulas are not used for this project.

The Kripke semantics evaluation function is shown in figure 4.8. This function is lengthy. Therefore, this figure shows only part of the function definition. The reader should note that the definition is defined for all Kripke structures as $\forall Oi Os M$.

The implementation of the "satisfies" property is shown in figure 4.9. The definition reads "f is true for the Kripke structure if and only if it evaluates to all worlds." The fact that is is defined for all Kripke structures $\forall M Oi Os$ makes it sound.
Figure 4.8: A partial description of the HOL implementation of the Kripke semantics (evaluation function). Image from *Certified Security by Design Using Higher Order Logic* [7]

\[
\text{[Efn\_def]}
\]
\[
\vdash (\forall \text{Oi Os M. Efn Oi Os M TT} = \mathcal{U}(\vdash \forall)) \land \\
(\forall \text{Oi Os M. Efn Oi Os M FF} = \{\}) \land \\
(\forall \text{Oi Os M p. Efn Oi Os M (prop p) = intpKS M p}) \land \\
(\forall \text{Oi Os M f.} \\
\text{Efn Oi Os M (notf f) = } \mathcal{U}(\vdash \forall) \text{ DIFF Efn Oi Os M f}) \land \\
(\forall \text{Oi Os M f \_ f\_2.} \\
\text{Efn Oi Os M (f\_1 andf f\_2) =} \\
\text{Efn Oi Os M f\_1} \cap \text{ Efn Oi Os M f\_2}) \land \\
(\forall \text{Oi Os M f \_ f\_2.} \\
\text{Efn Oi Os M (f\_1 orf f\_2) =} \\
\text{Efn Oi Os M f\_1} \cup \text{ Efn Oi Os M f\_2}) \land \\
(\forall \text{Oi Os M f \_ f\_2.} \\
\text{Efn Oi Os M (f\_1 impf f\_2) =} \\
\mathcal{U}(\vdash \forall) \text{ DIFF Efn Oi Os M f\_1} \cup \text{ Efn Oi Os M f\_2}) \land \\
(\forall \text{Oi Os M f \_ f\_2.} \\
\text{Efn Oi Os M (f\_1 eqf f\_2) =} \\
(\mathcal{U}(\vdash \forall) \text{ DIFF Efn Oi Os M f\_1} \cup \text{ Efn Oi Os M f\_2}) \cap \\
(\mathcal{U}(\vdash \forall) \text{ DIFF Efn Oi Os M f\_2} \cup \text{ Efn Oi Os M f\_1})) \land
\]

Figure 4.9: The HOL implementation of the "satisfies" property. Image from *Certified Security by Design Using Higher Order Logic* [7]

\[
\text{[sat\_def]}
\]
\[
\vdash \forall M \text{ Oi Os f. } (M, Oi, Os) \text{ sat } f \iff (\text{Efn Oi Os M f} = \mathcal{U}(\vdash \text{world}))
\]

This section described only the ACL and HOL implementation that are used in this project. The reader can find additional information in the *Certified Security by Design Using Higher Order Logic* [7].
Chapter 5

Secure State Machine Model

This section describes the secure state machine (SSM) and its implementation in HOL. It begins with a description of a state machine and then describes a secure state machine as a security-enhanced version of the state machine. This section concludes with a description of the HOL implementation of the secure state machine used for this UAV described in the previous chapter.

State Machines

A state machine is a transition model of a system that behaves in a well-defined way that is readily automated. There are various models of state machines. The model used for this project describes a state machine as a system with a finite number of states, commands (or inputs) to transition from one state to another, and an output the follows each transition. The state machine is modeled with next-state (NS) and next-output (NOut) functions. These functions define how the state machine should change states and outputs, respectively, given a specific command.
In addition, the state machine defines a configuration which consists of (1) the current state, (2) a list of commands (an input stream), and (3) a list of outputs (an output stream). The configuration also defines the state machines behavior. A configuration has the following form.

\[
\text{CFG} \\
\text{input} ::= \text{inputList} \\
\text{CURRENT\_STATE} \\
\text{outList}
\]

Where \(\text{CFG}\) is the constructor for the configuration type\(^1\), \(\text{input} ::= \text{inputList}\) is the input consed (prepended) to the remainder of the input stream (list), \(\text{CURRENT\_STATE}\) is the current state, and \(\text{outList}\) is the output stream (list).

A transition relation (TR) defines the state machines behavior based on the command (input stream), current configuration, and the next configuration. The format for the TR as follows.

\[
\text{TR} \text{ input} \\
\text{CFG} \\
\text{input} ::= \text{inputList} \\
\text{CURRENT\_STATE} \\
\text{outList} \\
\text{CFG} \\
\text{inputList} \\
(\text{NS input CURRENT\_STATE}) \\
(\text{NOut input CURRENT\_STATE })::\text{outList}
\]

The TR takes an input, an initial configuration, and a final configuration as inputs. It returns true if the final configuration results from applying the input to the initial configuration. Otherwise, it returns false.

\(^1\)Constructors are necessary for datatypes and used in HOL which is a strongly typed language. In essence, it announced that what follows is a list of the parameters for the CFG (configuration) datatype.
The second configuration results from the transition from the current state (or configuration) to the next state (or configuration). This transition is defined by the next-state (NS) and next-output (NOut) functions. In the second configuration, these are denoted as \textit{NS input CURRENT\_STATE} and \textit{NOut input CURRENT\_STATE}, respectively. Both of these functions take the current command (or input) and the current state and return the next state and next output, respectively. The input is the input from the initial configuration. The input has been "used-up" and is removed from the input stream in the second configuration. Thus, the input for the second configuration is just \textit{inputList}.

**Secure State Machines**

A secure state machine is a state machine with a security component attached. In particular, a secure state machine includes a security monitor which checks transitions for authentication and authorization. Figure 5.1 shows how the state machine relates to the secure state machine.

![Figure 5.1](image.png)

Figure 5.1: State machine versus secure state machine with a monitor. Image taken from \textit{Certified Security by Design Using Higher Order Logic} [7].

In contrast to a state machine, a secure state machine requires an entity to request a
transition from one state (or configuration) to another. If this entity (referred to as a principal) is not properly cleared, then the security monitor does not allow the transition. In particular, there are three types of actions that the monitor allows: (1) execute \((\text{exec})\) the transition, (2) trap \((\text{trap})\) the transition, or (3) discard \((\text{discard})\) the command (input). The monitor allows a transition to be executed if and only if the principal is both authenticated and authorized on a particular transition. The monitor traps a transition if the principal is authenticated but not authorized. The monitor discards a command (or input) if the principal is not authenticated. Note that if the principal is not authenticated then the monitor does not need to check if the principal is authorized. Thus, the command, rather than the transition, is discarded.

In addition to the next-state and next-output functions defined for the state machine, the secure state machine also defines an authentication function, a state-based authorization function, and a global authorization function. The authentication function returns true if the principal is authenticated. The state-based authorization function defines the state-based security (access-rights) policy. The global authorization function defines security (access-rights) for the entire secure state machine. With these new definitions, the configuration now has six components.

\[
\text{CFG}
\]

- \(\text{authenticationTest}\)
- \(\text{stateInterpretation}\)
- \(\text{securityContext}\)
- \(\text{input::inputList}\)
- \(\text{CURRENT\_STATE}\)
- \(\text{outList}\)

Where \(\text{authenticationTest}\), \(\text{stateInterpretation}\), and \(\text{securityContext}\) are the authentication function, state-based authorization function, and global authorization function, respectively.

The TR for the secure state machine functions the same as that for the state machine.
The only difference is that the configurations are for the secure state machine. The TR for the secure state machine is as follows.

\[ \text{TR (trType input)} \]

\[ \text{CFG} \]

\[ \text{authenticationTest} \]
\[ \text{stateInterpretation} \]
\[ \text{securityContext} \]
\[ \text{input::inputList} \]
\[ \text{CURRENT\_STATE} \]
\[ \text{outList} \]

\[ \text{CFG} \]

\[ \text{authenticationTest} \]
\[ \text{stateInterpretation} \]
\[ \text{securityContext} \]
\[ \text{inputList} \]
\[ (\text{NS (trType input) CURRENT\_STATE}) \]
\[ (\text{NOut (trType input) CURRENT\_STATE })::\text{output} \]

To enforce security, the monitor must interpret the meaning of the configuration using the access-control logic. It does this with a configuration interpretation function \((\text{CFGConfig})\). This function interprets the authentication and authorization functions using the rules of the ACL. It applies these rules to the initial configuration and generates a list of propositions that the configuration satisfies. The \(\text{CFGInterp} \) function is as follows.

\[ \text{ConfigInterp} \]

\[ (\text{M,Oi,Os}) \]

\[ \text{CFG} \]

\[ \text{AuthenticationTest inputOK} \]
\[ \text{StateInterpretation} \]
SecurityContext
input::inputList
CURRENT_STATE
outList

\[ \iff \]

\((M,O_i,O_s) \text{ satisfies (securityContext, input, and stateInterpretation)}\)

Where the \(\iff\) is the symbol for a biconditional (if and only if) and \((M,O_i,O_s)\) is a Kripke structure.\(^2\)

### The Parametrizable Secure State Machine in HOL

A parametrizable secure state machine is implemented in HOL for this project. To use this for a specific secure state machine, the secure state machine must define a next-state and next-output function, an authentication function, a state-based and global authorization function, and a list of principals, commands, and outputs. With these defined, the specific secure state machine can use the function and pre-proved theorems of the parametrizable secure state machine to ease the process of working with HOL.

The transition rules (TR rules) defined in HOL are shown in figure 5.2.

The interpretation of figure 5.2 follows from the discussion above. \(\text{elementTest}\) is defined for the specific secure state machine. It returns a list of true/false values for each user-defined principal. \(\text{authenticationTest}\) returns the conjunction of all the list elements. \(\text{context}\) is the same as \(\text{securityContext}\) defined above.

For the parametrizable secure state machine, the TR rule to execute a transition is shown in figure 5.3. The use of Kripke structures allows HOL to prove that any

\(^2\)The Kripke structure is not defined because the rules for satisfies apply to all Kripke structures.
SSM behavior is defined inductively by three rules.

\[
\text{Execute} \quad \frac{\text{authenticationTest elementTest } x \quad \text{CFGInterp } (M, O_i, O_b) \quad \text{Config}}{\text{Config}_{\text{exec } (\text{inputList } x)} \rightarrow \text{Config}_e} \\
\text{Trap} \quad \frac{\text{authenticationTest elementTest } x \quad \text{CFGInterp } (M, O_i, O_b) \quad \text{Config}}{\text{Config}_{\text{trap } (\text{inputList } x)} \rightarrow \text{Config}_t} \\
\text{Discard} \quad \frac{\neg \text{authenticationTest elementTest } x}{\text{Config}_{\text{discard } (\text{inputList } x)} \rightarrow \text{Config}_d}
\]

where,

\[
\begin{align*}
\text{Config} &= \text{CFG elementTest stateInterp context } (x :: \text{ins}) \text{ s outs} \\
\text{Config}_e &= \text{CFG elementTest stateInterp context ins} \\
&(\text{NS s (exec (inputList } x))) \quad (\text{Out s (exec (inputList } x)) :: \text{outs}) \\
\text{Config}_t &= \text{CFG elementTest stateInterp context ins} \\
&(\text{NS s (trap (inputList } x))) \quad (\text{Out s (trap (inputList } x)) :: \text{outs}) \\
\text{Config}_d &= \text{CFG elementTest stateInterp context ins} \\
&(\text{NS s (discard (inputList } x))) \quad (\text{Out s (discard (inputList } x)) :: \text{outs})
\end{align*}
\]

Figure 5.2: Transition rules in HOL. Image taken from Certified Security by Design Using Higher Order Logic [7]

inference rule derived from this is sound using the ACL. On the top line, TR takes a Kripke structure and a command. The command is preceded by the transition type (trType) exec. The next line is the initial configuration. The subsequent three lines are the final configuration. Following the biconditional symbol $\iff$ is the application of the security monitor. This is the authentication function and (conjunction) the configuration interpretation function applied to the initial configuration. The rule states that the transition should be executed if and only if the principal is authenticated and the initial configuration satisfies the $\text{CFGInterp}$ relation. This later relation allows HOL to transform the initial configuration into a list that the ACL can use to draw conclusions about authentication. The rule for trap is similar.

The TR rule for the exec is used to prove that a transition should be executed if and only if the request is authenticated and authorized. This rule is named
\[ \vdash \text{TR} \left(M, Oi, Os\right) \left(\text{exec \ (inputList \ x)}\right) \]

\[ (\text{CFG elementTest stateInterp context (}x::\text{ins}) s \ \text{outs}) \]

\[ (\text{CFG elementTest stateInterp context ins} \]

\[ (\text{NS} \ s \ (\text{exec \ (inputList \ x)}) \]

\[ (\text{Out} \ s \ (\text{exec \ (inputList \ x)})::\text{outs})) \iff \]

\[ \text{authenticationTest \ elementTest \ x} \land \]

\[ \text{CFGInterpret} \ \left(M, Oi, Os\right) \]

\[ (\text{CFG elementTest stateInterp context (}x::\text{ins}) s \ \text{outs}) \]

Figure 5.3: TR rule for the exec transition rule in HOL.

\textit{TR\_exec\_cmd\_rule}. It is shown in figure 5.4.

The top line states that this theorem should be true for any authentication function \textit{(elementTest)}, global authorization function \textit{(context)}, state-based authorization function \textit{(stateInterp)}, input \textit{(x)}, input stream \textit{(ins)}, state \textit{(s)}, and output stream \textit{(outs)}. The antecedent of the first implication (preceeding the first \(\Rightarrow\) symbol) applies the \textit{CFGInterp} function to the initial configuration. The conclusion (and the antecedent of the second implication) extracts the propositions (or requests) from the input \textit{x}. The conclusion of the second implication applies the TR relation shown in figure 5.2. This is parametrized by the next-state \textit{(NS)} and next-output \textit{(NOut)} functions and the Kripke structure. The conclusion of the biconditional is conjunction of the authentication function and the \textit{CFGInterp} function applied to the initial configuration. The theorem proves that the configuration interpretation implies the input which implies the transition is executed if and only if the input is authenticated and authorized. The rule for trap is similar.

The TR rule for discard leads directly to the theorem \textit{TR\_discard\_cmd\_rule}. The
\[ \vdash \forall \text{elementTest context stateInterp } x \text{ ins } s \text{ outs} . \]

\[
(\forall M Oi Os .

\text{CFGInterpret } (M, Oi, Os)

(CFG \text{ elementTest stateInterp context } (x :: \text{ins}) s \text{ outs}) \Rightarrow

(M, Oi, Os) \text{ satList propCommandList } x) \Rightarrow

\forall NS Out M Oi Os .

\text{TR } (M, Oi, Os) \text{ (exec } (\text{inputList } x ) )

(\text{CFG elementTest stateInterp context } (x :: \text{ins}) s \text{ outs})

(\text{CFG elementTest stateInterp context ins}

(NS s (exec (\text{inputList } x)))

(Out s (exec (\text{inputList } x))::\text{outs})) \iff

\text{authenticationTest elementTest } x \land

\text{CFGInterpret } (M, Oi, Os)

(\text{CFG elementTest stateInterp context } (x :: \text{ins}) s \text{ outs}) \land

(M, Oi, Os) \text{ satList propCommandList } x
\]

Figure 5.4: TR_exec_cmd_rule for the exec transition rule in HOL.

rules is shown in figure 5.5.
\[ \vdash \text{TR} (M, O_i, O_s) \ (\text{discard } (\text{inputList } x)) \]

\[
\begin{align*}
& (\text{CFG } \text{elementTest } \text{stateInterp context } (x::\text{ins}) \ s \ \text{outs}) \\
& (\text{CFG } \text{elementTest } \text{stateInterp context } \text{ins} \\
& \quad (\text{NS } s \ (\text{discard } (\text{inputList } x)))) \\
& \quad (\text{Out } s \ (\text{discard } (\text{inputList } x))::\text{outs}) \iff \\
& \neg \text{authenticationTest } \text{elementTest } x
\end{align*}
\]

Figure 5.5: TR_discard_cmd_rule for the discard transition rule in HOL.
Part II

STORM on Patrol Base Operations
Chapter 6

STPA/STPA-Sec on Patrol Base Operations

This chapter applies the STPA/STPA-Sec analysis to the patrol base operations. The patrol base operations are analyzed at a high level of abstraction and only the actions of the Platoon Leader (PL) and the platoon as a whole are considered. The limited scope of the analysis serves to demonstrate the STPA component of System-Theoretic and Technical Operational Risk Management (STORM) on the system of interest.

This chapter describes each of the four steps of the STPA/STPA-Sec analysis. The results are discussed in the text and presented in tables. For a review on the four steps read section 3.0.2.

6.1 Step 1: Define The Purpose of The Analysis

Stakeholders At the highest level, the stakeholders are the U.S. Government and the Citizens of the United States. Below that are the U.S. Army and the U.S. Army
Rangers. Depending on the mission, other nations and their citizens may also hold a stake in the activities. The enemy is also a stakeholder with an adversarial agenda.

**Purpose**  The purpose of this thesis is to demonstrate STPA (STORM) on a non-automated, human-centered system. Therefore, the analysis need only be conducted to the extent that all the steps are explored at least once. STPA is an iterative process, thus an analysis for a different purpose would entail refinements as needed.

**Organization FMA**  The two most immediate organizations for the patrol base operations are the U.S. Army Rangers and the U.S. Army. Their goals are stated in table 6.1.

<table>
<thead>
<tr>
<th>Organization Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S. Army</td>
</tr>
<tr>
<td>&quot;The U.S. Army’s mission is to fight and win our Nation’s wars by providing prompt, sustained land dominance across the full range of military operations and spectrum of conflict in support of combatant commanders.&quot; [18]</td>
</tr>
<tr>
<td>U.S. Army Rangers</td>
</tr>
<tr>
<td>&quot;The Rangers’ primary mission is to engage the enemy in close combat and direct-fire battles. This mission includes direct action operations, raids, personnel and special equipment recovery, in addition to conventional or special light-infantry operations.&quot; [9]</td>
</tr>
</tbody>
</table>

Table 6.1: Organization goals.
**Patrol Base Operations FMA**  The functional mission analysis for the patrol base operations is shown in table 6.2. This describes what the patrol base operations are, how they are conducted (broadly), and why they are conducted.

<table>
<thead>
<tr>
<th>What</th>
<th>establish a security perimeter when a squad or platoon halts for an extended period of time</th>
</tr>
</thead>
<tbody>
<tr>
<td>How</td>
<td>planning, reconnaissance, security, control, and common sense</td>
</tr>
<tr>
<td>Why</td>
<td>avoid detection:</td>
</tr>
<tr>
<td></td>
<td>• hide a unit during a long,</td>
</tr>
<tr>
<td></td>
<td>• detailed reconnaissance;</td>
</tr>
<tr>
<td></td>
<td>• perform maintenance on weapons,</td>
</tr>
<tr>
<td></td>
<td>• equipment, eat, and rest;</td>
</tr>
<tr>
<td></td>
<td>• plan and issue orders;</td>
</tr>
<tr>
<td></td>
<td>• reorganize after infiltrating an enemy area;</td>
</tr>
<tr>
<td></td>
<td>• establish a common base from which to execute several consecutive or concurrent operations.</td>
</tr>
</tbody>
</table>

Assumptions about The System  This analysis relies on the following assumptions about the model of patrol base operations.

- Mission is decided by higher up leaders and is provided as an input to the patrol base operations.
- Mission is not changeable by the patrol base itself (exceptions are mission-specific).
- Soldiers are deemed fit for duty before the mission commences. (There may be exceptions.)
- Soldiers are battle ready, but may not be fully mission capable (i.e., soldiers may require additional equipment and preparation that is mission-specific and determined after the mission is received).
- The patrol base operations begin when the platoon (or patrol) leader receives order that there will be a mission.
- The patrol base operations end when the patrol returns to the larger unit and has completed any debriefing required by the mission.
- Other inputs to the system are intelligence from the larger unit (higher-up HQ), intelligence from other units (if applicable), and intelligence gathered during the mission from the mission itself.
- Feedback is received from the platoon regarding carrying out orders and from the operations themselves regarding completion of tasks, etc.
- Adversary also may input information to the platoon leader (by interfering with communications from higher-up HQ), to the platoon soldiers (by interfering with intra-patrol communications), and to the operations themselves (by disrupting the operations in various forms, including contact).
**System Entities** The following entities are defined in the U.S. Ranger Handbook for the patrol base operations.

- HQ: PL, PSG, Medic, FO, RTO, HWSQ
- Squad Leaders
- Fire Team Leaders
- Buddy Team
- Soldiers
- Adversary (enemy)

The platoon leader (Platoon Leader – PL) is the only entity specifically referred to in this limited-scope analysis. The platoon sergeant (PSG) is the assistant to the platoon leader. The patrol base headquarters (HQ) consists of the PL, PSG, Medic, forward observer (FO), radio telephone operator (RTO), and the heavy weapons squad leader (HWSQL). Below the HQ are four squad leaders. Each squad is comprised of two fire teams. Each fire team is composed of two or more buddy teams. Each buddy team is composed of two soldiers (buddies).

---

1This is only true for platoon sized patrol base operations. Squad-sized patrol base operations consist of only one squad.
**Accidents/Losses**  Accidents/losses for the patrol base operations are shown in table 6.3.

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 MIA, KIA, WIA, CIA</td>
<td></td>
</tr>
<tr>
<td>L2 Wrong mission (i.e., mission not clearly communicated causing wrong</td>
<td></td>
</tr>
<tr>
<td>objectives accomplished)</td>
<td></td>
</tr>
<tr>
<td>L3 Mission Failure (i.e., right mission, but objectives not achieved)</td>
<td></td>
</tr>
<tr>
<td>L4 Negative publicity/exposure/unwanted attention</td>
<td></td>
</tr>
<tr>
<td>L5 Equipment loss/damage/capture by enemy</td>
<td></td>
</tr>
<tr>
<td>L6 Civilian casualties/disruption to local population</td>
<td></td>
</tr>
<tr>
<td>L7 Insufficient communications with high-up HQ (i.e., causing inability for</td>
<td></td>
</tr>
<tr>
<td>high-ups to monitor and coordinate actions with other missions and resources</td>
<td></td>
</tr>
<tr>
<td>or to keep key stakeholders informed of progress.)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: System accidents/losses.

Note the following definitions: MIA = missing in action; KIA = killed in action; WIA = wounded in action; CIA = captured in action.
**System-level Hazards/Vulnerabilities And Constraints**  
Table 6.4 describes the system-level hazards/vulnerabilities and associated system-level constraints. The system-level hazards/vulnerabilities are described in the second column. The associated accidents/losses for each hazard are shown in the next column. These are linked to system-level constraints in the last column.

<table>
<thead>
<tr>
<th>System-level Hazards/Vulnerabilities</th>
<th>Accidents/Losses</th>
<th>System-level Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 Insufficient planning</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC1 Plan sufficiently</td>
</tr>
<tr>
<td>H2 Insufficient reconnaissance/intelligence</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC2 Reconnoiter sufficiently</td>
</tr>
<tr>
<td>H3 Insufficient security</td>
<td>L1, L3, L4, L5, L6</td>
<td>SC3 Provide sufficient security</td>
</tr>
<tr>
<td>H4 Insufficient control</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC4 Maintain adequate control</td>
</tr>
<tr>
<td>H5 Insufficient common sense</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC5 Use common sense</td>
</tr>
<tr>
<td>H6 Insufficient leadership</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC6 Establish sufficient leadership</td>
</tr>
<tr>
<td>H7 Insufficient communication with higher-HQ</td>
<td>L1, L2, L3, L4, L5, L6, L7</td>
<td>SC7 Maintain sufficient communications with higher-HQ</td>
</tr>
<tr>
<td>H8 Insufficient haste</td>
<td>L1, L3</td>
<td>SC8 Maintain appropriate pace</td>
</tr>
</tbody>
</table>

Table 6.4: System-level hazards/vulnerabilities and constraints.

A more in-depth analysis would return to this table and generate refined hazards/vulnerabilities and constraints. For example, H1 could be refined to H1.1 insufficient reconnaissance, H1.2 mission objectives not clear, H1.3 insufficient time for completing plan, etc.. For the constraints, SC1.1 reconnoiter efficiently, SC1.2 confirm mission objectives, SC1.3 use time efficiently, etc..
6.2 Step 2: Model The Control Structure

This section first divides the patrol base operations into phases. It then describes the system controllers and their process models. It then describes the control actions for each controller. Finally, it presents a diagram of the functional control model.

**Phases of The Patrol Base Operations**  Figure 6.5 shows the patrol base operations divided into six phases. This is a simplified first approximation of the patrol base operations and is very abstract. For the purposes of this analysis, transitions among phases of the operations are indicated by the Platoon Leader who is responsible for the success of the mission as a whole.

![Patrol Base Phases (REVISED)](image)

Table 6.5: Scenarios for UCA E1A3. Diagram by Captain Jesse Nathaniel Hall.
Controllers And Process Models  The controllers and their process models are shown in table 6.6. There are three controllers: the Platoon Leader, the platoon, and the adversary (enemy). The Platoon Leader is the top controller. She issues orders to the subordinates (the platoon) based on her knowledge of the status and policy of the operations. The platoon authenticates the Platoon Leader and either executes the orders or discards the commands.

Each controller has one or more models of the patrol base operations that governs her decisions and actions. Each model has a variable that can take on one of several values. For example, the Platoon Leader has a model of the Patrol Base Status. The status is the State of the system. For example, the State could be PLAN_PB or MOVE_TO_ORP.

The Platoon Leader’s Policy model defines which conditions and which commands are authorized. For example, for most instances of the patrol base operations, the planning phase should be completed before moving to the objective rally point (ORP).² Therefore, the Platoon Leaders policy would include the condition that planning phase

<table>
<thead>
<tr>
<th>Controller</th>
<th>Model</th>
<th>Variables</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platoon Leader</td>
<td>Patrol Base Status</td>
<td>State</td>
<td>PLAN_PB, MOVE_TO_ORP, CONDUCT_ORP, MOVE_TO_PB, CONDUCT_PB, COMPLETE_PB, ABORT_PB</td>
</tr>
<tr>
<td>Platoon</td>
<td>Authentication Model</td>
<td>Authenticated</td>
<td>True, False</td>
</tr>
<tr>
<td>Adversary</td>
<td>Threat Model</td>
<td>To be determined</td>
<td>to be determined</td>
</tr>
</tbody>
</table>

Table 6.6: Controllers and process model.

²The most likely variation is for the platoon to move to the ORP and then receive orders. Regardless, it is unlikely that the platoon will move to the patrol base operations before completing planning.
complete implies move to the ORP. The Authorization variable for the Policy model would be true if the phase is PLAN_PB and this phase is complete.

The platoon’s Authentication Model defines how the platoon authenticates commands. For example, the platoon may authorize commands by visual confirmation of the leader issuing the command. Then, whenever the Platoon Leader issues the command in the line of sight of the platoon, the Authentication variable would take the value True. The platoon may also authenticate the platoon leader via a password. To simplify the analysis for this thesis, the platoon leader is assumed to be authenticated by recognition by the platoon.

The adversary’s process model is the threat model. It is determined from METT-TC\(^3\) analysis. It may or may not be known or complete. The more known and complete the model, the more capable the platoon is at anticipating threats from the enemy. But, knowledge of a specific enemy should not guide the system design. The design should build a defensible mission that aims to protect stakeholder assets regardless of the enemy’s capabilities (if possible). To simplify this analysis, the threat model is not explored.

---

\(^3\)mission, enemy, terrain and weather, troops and support available-time available, and civil considerations factors.
**Control Actions** Each controller has its own set of actions (commands, etc.) that it can issue. The control actions for each principal are listed below.

- **Platoon Leader**
  - PL says planComplete (same as crossLD)
  - PL says conductORP
  - PL says moveToPB
  - PL says completePB
  - PL says reactToContact
  - PL says returnToBase
  - PL says changeMission

- **Platoon**
  - exec(crossLD) (same as planComplete)
  - exec(conductORP)
  - exec(moveToPB)
  - exec(completePB)
  - exec(reactToContact)
  - exec(returnToBase)
  - exec(changeMission)
  - discard(anyCommand)

For example, the Platoon Leader may determine (based on policy, etc.), that the planning phase is complete. She may then indicate this with the statement *PL says planComplete*.\(^4\) Similarly, the platoon may respond to this command with

\(^4\)crossLD = cross line of discrimination. Depending on the model, this would occur after planning. It would transition the patrol into the MOVE_TO_ORP phase of the operations. A description of this works is reserved for chapter 7.
exec(planComplete). If the platoon is in the PLAN_PB phase, it would then move to the next phase MOVE_TO_ORP.

The discard(anyCommand) indicates that the command, whatever it is, should be discarded.

**Functional Control Structure**  The functional control structure diagrams the command and control structure with feedback. It is shown in figure 6.1.

![Control structure for patrol base operations.](image)

The blue background encapsulates the command and control structure of the operations. The yellow background encapsulates the access-control structures. The green is the adversarial realm with both command and control and access-control capabilities.
Downward pointing arrows indicate the direction of commands and upward pointing arrows indicate feedback. The adversary is an exception. These arrows are directed out towards the controllers (PL and platoon) and the controlled actions (patrol base operations). Feedback to the adversary is not modeled. However, modeling feedback may shine light on avenues of attack or counter-attack on the adversary. For example, reacting to contact with the enemy would be modeled as a feedback from the patrol base operations to the adversary. Feedback from the platoon or platoon leader could be deceptive counterintelligence messages misleading the enemy as to the patrol base operations locations or objectives. Modeling this feedback would provide another analysis consideration for the analyst.

The Platoon Leader is the legitimate controller in the command and control structure. She sends commands to the platoon. The commands are shown in the grey box. The Platoon Leader receives feedback from the platoon (i.e, discard command, confirm orders, etc.). The Platoon Leader also receives orders or intelligence from higher-up headquarters, the operations, and potentially from other sources.

The platoon receives orders from the Platoon Leader and either executes them or discards them. Execution of an order implies that the Platoon Leader’s command was authenticated. Discarding means that the command was not authenticated. The platoon also receives information from the operations as to what phase it is in, etc..

The adversary can interfere with the operations in numerous ways. It can attack the Platoon Leader or platoon. It can send conflicting messages, such as send unauthenticated commands in the hope that the platoon or Platoon Leader will authenticate it. The adversary can attack or sabotage the platoon. The capabilities of the adversary to interrupt the operations should be analyzed as soon as possible to develop an accurate threat model.
6.3 Step 3: Identify Unsafe Control Actions (UCAs)

This section determines unsafe control actions (unsafe control actions (UCA)) for the patrol base operations. It uses the Thomas Method\textsuperscript{5} to identify UCAs. This method considers all possible combinations of values of the process model variables.

All possible process model values are analyzed for each control action. This means that for any control action the possible combinations of process model values is $7 \times 2 \times 2 = 28$.

There are 7 possible control actions (commands) for the Platoon Leader and $2 \times 7$ possible control actions for the platoon. In total, there are $7 + 2 \times 7 = 21$ possible control structures. To analyze each control structure requires $21 \times 28 = 588$ possible UCAs.

Fortunately, many UCAs are similar or can be grouped. Nevertheless, to limit this analysis for its demonstrative purposes only three control actions are analyzed: \textit{PL says planComplete}, \textit{exec(planComplete)}, and \textit{discard(anyCommand)}. Notice that the last control action covers all 7 discard commands because discarding any command is the same for all commands.

The UCAs are analyzed using the tables 6.7, 6.8, and 6.9 in the next section. In each of these tables, each control action is analyzed for each combination of process model value and for each of the four types of UCAs. The four UCAs described in section 3.0.2 of the previous chapter are repeated in the list below.

- Not applying the control action

\textsuperscript{5}John P. Thomas, PhD is coauthor with Nancy Leveson, PhD on the STPA Handbook. The Thomas Method was noted in one of my readings on the subject. But I have not been able to find a citation for the paper. It was noted only in one sentence which referred to the systematic way in which the UCAs are documented in this thesis.
• Applying the wrong control action

• Applying the control action in the wrong order

• Applying the control action for two long or not long enough.

To make the tables more manageable, for each combination of control action and process model value the table indicates if there is an UCA by marking the appropriate box with a code. The code will be used in step 4 to describe scenarios. If the box is empty, then no UCA is determined for those set of values.
### 6.3.1 Unsafe Control Actions (UCAs): Thomas Model

**PL says planComplete**  Table 6.7 analyzes the UCAs for the control action *PL says planComplete*. Indexing the UCAs with keys such as "P1A1" and "P1B1" helps to manage the size and complexity of the tables. These keys are referenced in step 4 of the analysis where the scenarios for each UCA are examined and explained in detail.

<table>
<thead>
<tr>
<th>Control Actions</th>
<th>Process Model Values</th>
<th>Unsafe Control Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL says planComplete Platoon Leader Platoon</td>
<td>PLAN_PB</td>
<td>P1B1 P1C1</td>
</tr>
<tr>
<td>yes yes</td>
<td>MOVE_TO_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPLETE_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABORT_PB</td>
<td></td>
</tr>
<tr>
<td>yes no</td>
<td>PLAN_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPLETE_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABORT_PB</td>
<td></td>
</tr>
<tr>
<td>no yes</td>
<td>PLAN_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPLETE_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABORT_PB</td>
<td></td>
</tr>
<tr>
<td>no no</td>
<td>PLAN_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_ORP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE_TO_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDUCT_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPLETE_PB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABORT_PB</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: Unsafe control actions UCAs for control action "PL says planComplete."
Table 6.8 analyzes the UCAs for the control action `exec(planComplete)`.

<table>
<thead>
<tr>
<th>Control Actions</th>
<th>Process Model Values</th>
<th>Unsafe Control Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exec(planComplete)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PL says planComplete</strong></td>
<td><strong>Patrol Base Ops</strong></td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td><code>PLAN_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>COMPLETE_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ABORT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>E1A1</code></td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td><code>PLAN_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>COMPLETE_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ABORT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>E1A2</code></td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td><code>PLAN_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>COMPLETE_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ABORT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>E1A3</code></td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td><code>PLAN_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_ORP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MOVE_TO_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CONDUCT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>COMPLETE_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ABORT_PB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>E1A4</code></td>
</tr>
</tbody>
</table>

Table 6.8: Unsafe control actions UCAs for control action "exec(planComplete)."
**discard(anyCommand)**  Table 6.9 analyzes the UCAs for the control action `discard(anyCommand)`.

<table>
<thead>
<tr>
<th>Control Actions</th>
<th>Process Model Values</th>
<th>Unsafe Control Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>discard</strong></td>
<td>Platoon</td>
<td>Patrol Base Ops</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 6.9: Unsafe control actions UCAs for control action "discard(anyCommand)."
6.4 Step 4: Identify Loss Scenarios

This section describes the scenarios associated with each UCA identified in the previous step.

6.4.1 Scenarios

The scenarios are described in a series of tables. The top row of each table indicates which control action is analyzed. The next row describes the process model variable values. It may also contain other useful information.

The first column, labeled "Scenarios", identifies the UCA by its code (from the tables in the previous section. It also describes the unsafe scenario. The next column, labeled "Associated Causal Factors", describes factors that could cause the unsafe scenario. The third column describes the rationale for mitigating or avoiding the causal factors.

The last two columns contain lists of alternating black and blue entries. Alternating with these colors associates an Associated Causal Factor in one column with a Rationale in the adjacent column. For example, the top blue Associated Causal Factor corresponds to the top blue Rationale in the next column. The next black entry Associate Causal Factor corresponds to the next black entry in the Rationale column, and so on.
Pl says planComplete

The next set of tables describes scenarios for the PL says planComplete control action.

P1B1/P1C1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes
Tables 6.10 and 6.11 present the scenarios for the P1B1 and P1C1 UCAs.
Table 6.10: Scenarios for UCA P1B1. (GOTWA = "(G) where I’m Going, (O) Others I’m taking, (T) Time of my return, (W) What to do if I don’t return, (A) Actions to take if I’m hit or Actions to take if you’re hit (5 point contingency plan)" [10]).
<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1B1 and P1C1. Adversary interferes with communications (i.e., blocks radio signals).</td>
<td>● Enemy interfering with communications</td>
<td>● Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares).</td>
</tr>
<tr>
<td>P1C1. PL does not complete plan with sufficient haste. H1,H5,H6,H8.</td>
<td>● PL does not plan to complete mission with sufficient haste.</td>
<td>● Establish standards for length of time needed to plan, train leaders, rehearse.</td>
</tr>
<tr>
<td></td>
<td>● Circumstances require planning to be completed at another time.</td>
<td>● Have a contingency plan for what to do until plan is complete.</td>
</tr>
<tr>
<td></td>
<td>● Enemy disruptions to operations cause plan completion to be delayed or aborted.</td>
<td>● Analyze enemy. Establish plan for early contact.</td>
</tr>
<tr>
<td></td>
<td>● METT-TC considerations interfere with completion of plan.</td>
<td>● Gather intel for METT-TC before mission. Establish contingency plan.</td>
</tr>
<tr>
<td></td>
<td>● Insufficient intel/recon to complete plan.</td>
<td>● Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.</td>
</tr>
</tbody>
</table>

Table 6.11: Scenarios for UCA P1B1/P1C1. (METT-TC = mission, enemy, terrain, troops available, time, and civilian considerations [11]).
**P1A1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes**  
Table 6.12 presents the scenarios for the P1A1 UCA.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
</table>
| P1A1, PL never began or completed the plan. H1,H4,H6. | • Circumstances require planning to be completed at another time.  
• Enemy disruptions to operations cause plan completion to be delayed or aborted.  
• METT-TC considerations interfere with completion of plan.  
• Insufficient intel/recon to complete plan.  
• PL incapacitated.  
• Change of leadership without sufficient transfer of command. | • Have a contingency plan for what to do until plan is complete.  
• Analyze enemy. Establish plan for early contact.  
• Gather intel for METT-TC before mission. Establish contingency plan.  
• Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.  
• Establish SOP for transfer of command. GOTWA.  
• Establish SOP for transfer of command. GOTWA. |
| P1A1, Need to re-plan H3,H4,H6.       | • Contingency plan not established early in mission.  
• Higher-up HQ presents new intel or new mission objectives.  
• Enemy disruptions to operations make previous plan obsolete.  
• METT-TC considerations interfere with completion of plan.  
• New intel/recon requires significant change in plan.  
• Change of leadership without sufficient transfer of command. | • Establish contingency plan early in mission.  
• Contingency plan (training, SOP) for re-planning during mission.  
• Analyze enemy. Establish plan for early contact.  
• Gather intel for METT-TC before mission. Establish contingency plan.  
• Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.  
• Establish SOP for transfer of command. GOTWA. |

Table 6.12: Scenarios for UCA P1A1.
P1A2: state = any state, Authorized = Yes, Authenticated = No  

Table 6.13 presents the scenarios for the P1A2 UCA.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1A2. Adversary issues command. H1,H3,H4,H5,H6.</td>
<td>• Adversary has commandeered (lost or stolen) communications equipment.</td>
<td>• Secure equipment.</td>
</tr>
<tr>
<td></td>
<td>• Adversary is otherwise attempting to issue commands to platoon.</td>
<td>• Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares).</td>
</tr>
<tr>
<td>P1A2. Platoon does not recognize PL because of communications equipment failure. H1,H3,H4,H5,H6.</td>
<td>• Equipment not checked before mission.</td>
<td>• Check equipment prior to mission.</td>
</tr>
<tr>
<td></td>
<td>• Wrong equipment.</td>
<td>• Choose communication devices that will satisfy mission needs.</td>
</tr>
<tr>
<td></td>
<td>• Battery dead, equipment fails in service.</td>
<td>• Know operating distances, etc., of equipment.</td>
</tr>
<tr>
<td></td>
<td>• Equipment damaged or not functioning while on mission.</td>
<td>• Bring back-up communication equipment, batteries, antenna.</td>
</tr>
<tr>
<td></td>
<td>• Lack of operational know how.</td>
<td>• Tools and know-how to troubleshoot and repair equipment.</td>
</tr>
<tr>
<td></td>
<td>• Platoon not all on same channel.</td>
<td>• Train soldiers on equipment use.</td>
</tr>
<tr>
<td></td>
<td>• Terrain interference on transmission.</td>
<td>• Communicate communication channel to subordinates.</td>
</tr>
<tr>
<td></td>
<td>• Communications equipment cannot otherwise be used.</td>
<td>• Move to a better position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Contingency communications plan for equipment failure.</td>
</tr>
<tr>
<td>P1A2. Platoon does not recognize PL for other reasons. H1,H3,H4,H5,H6.</td>
<td>• PL incapacitated.</td>
<td>• Establish SOP for transfer of command. GOTWA.</td>
</tr>
<tr>
<td></td>
<td>• Change of leadership without sufficient transfer of command.</td>
<td>• Establish SOP for transfer of command. GOTWA.</td>
</tr>
<tr>
<td></td>
<td>• Mutiny!</td>
<td>• Discourage mutiny! Foster confidence in leadership. Vet leaders.</td>
</tr>
<tr>
<td></td>
<td>• No line-of-sight for visual communications signals.</td>
<td>• Move within line-of-sight. Contingency communications for no line-of-sight. Animal calls, radios, etc.</td>
</tr>
</tbody>
</table>

Table 6.13: Scenarios for UCA P1A2.
**P1A3:** state = any state, Authorized = No, Authenticated = Yes  
Table 6.14 presents the scenarios for the P1A3 UCA.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
</table>
| P1A3. PL indicates planning is complete when it is not. H1,H2,H4. | • Inadequate leadership.  
• Change of leadership without sufficient transfer of command.  
• Mutiny/rogue leadership. | • Vet leadership.  
• Establish SOP for transfer of command. GOTWA.  
• Discourage mutiny! Foster confidence in leadership. Vet leaders. |

Table 6.14: Scenarios for UCA P1A3.

**P1A4:** state = any state, Authorized = No, Authenticated = No  
Table 6.15 presents the scenarios for the P1A4 UCA.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
</table>
| P1A4. Adversary issues command. H1,H3,H4,H5,H6. | • Adversary has commandeered (lost or stolen) communications equipment.  
• Adversary is otherwise attempting to issue commands to platoon. | • Secure equipment.  
• Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares). |

Table 6.15: Scenarios for UCA P1A4.
The next set of tables describes scenarios for the `exec(planComplete)` control action.

**E1B1/E1C1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes**

Table 6.16 presents the scenarios for the E1B1 and E1C1 UCAs.

```
<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1B1. Failure to act when action is required.</td>
<td>Multin. Platoon does not follow orders.</td>
<td>Discourage mutiny. Foster confidence in leadership.</td>
</tr>
<tr>
<td></td>
<td>PL overloaded/too much work for one person.</td>
<td>Limit tasks to a manageable number.</td>
</tr>
<tr>
<td></td>
<td>Platoon is incapacitated and unable to follow orders.</td>
<td>Contingency plan for handling incapacitated soldiers and re-assigning responsibilities.</td>
</tr>
<tr>
<td></td>
<td>Perceived conflict with ROE.</td>
<td>Enforce ROE. Establish procedures for ROE problems (i.e., change of command, etc.).</td>
</tr>
<tr>
<td></td>
<td>Assigned tasks are not clear.</td>
<td>Foster confidence in leadership.</td>
</tr>
<tr>
<td></td>
<td>METT-TC conditions interfere with execution of orders.</td>
<td>Assign tasks, clarify, and rehearse.</td>
</tr>
<tr>
<td>E1B1 and E1C1. Adversary interferes with operations (i.e., engages platoon, etc.).</td>
<td>Contact with enemy.</td>
<td>Conduct enemy analysis. Develop contingency plan for operations, contact with enemy, etc.</td>
</tr>
<tr>
<td></td>
<td>Sabotage of plans.</td>
<td>Conduct enemy analysis.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contingency plan.</td>
</tr>
<tr>
<td>E1C1. exec planComplete not provided soon enough.</td>
<td>METT-TC conditions interfere with execution of orders.</td>
<td>Gather intel for METT-TC before mission. Establish contingency plan.</td>
</tr>
<tr>
<td></td>
<td>PL overloaded/too much work for one person.</td>
<td>Limit tasks to a manageable number.</td>
</tr>
<tr>
<td></td>
<td>Lack of discipline.</td>
<td>Train soldiers to be disciplined.</td>
</tr>
</tbody>
</table>
```

Table 6.16: Scenarios for UCAs E1B1 and E1C1.
E1A1: state \( \neq \) PLAN_PB, Authorized = Yes, Authenticated = Yes  

Table 6.17 present the scenarios for the E1A1 UCAs.

### Table 6.17: Scenarios for UCA E1A1.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1A1. Planning never began or completed.</td>
<td>- Circumstances require planning to be completed at another time.</td>
<td>- Have a contingency plan for what to do until plan is complete.</td>
</tr>
<tr>
<td></td>
<td>- Enemy disruptions to operations cause plan completion to be delayed or aborted.</td>
<td>- Analyze enemy. Establish plan for early contact.</td>
</tr>
<tr>
<td></td>
<td>- METT-TC considerations interfere with completion of plan.</td>
<td>- Gather intel for METT-TC before mission. Establish contingency plan.</td>
</tr>
<tr>
<td></td>
<td>- Insufficient intel/recon to complete plan.</td>
<td>- Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.</td>
</tr>
<tr>
<td></td>
<td>- P1 incapacitated.</td>
<td>- Establish SOP for transfer of command. GOTWA.</td>
</tr>
<tr>
<td></td>
<td>- Change of leadership without sufficient transfer of command.</td>
<td>- Establish SOP for transfer of command. GOTWA.</td>
</tr>
<tr>
<td></td>
<td>- Higher-up HQ presents new intel or new mission objectives.</td>
<td>- Contingency plan (training, SOP) for re-planning during mission.</td>
</tr>
<tr>
<td></td>
<td>- Enemy disruptions to operations make previous plan obsolete.</td>
<td>- Analyze enemy. Establish plan for early contact.</td>
</tr>
<tr>
<td></td>
<td>- METT-TC considerations interfere with completion of plan.</td>
<td>- Gather intel for METT-TC before mission. Establish contingency plan.</td>
</tr>
<tr>
<td></td>
<td>- New intel/recon requires significant change in plan.</td>
<td>- Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.</td>
</tr>
<tr>
<td></td>
<td>- Change of leadership without sufficient transfer of command.</td>
<td>- Establish SOP for transfer of command. GOTWA.</td>
</tr>
</tbody>
</table>
E1A2: state = any state, Authorized = Yes, Authenticated = No  

Table 6.18 present the scenarios for the E1A2 UCAs.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E1A2. Adversary issues command. H1,H3,H4,H5,H6.</strong></td>
<td>- Adversary has commandeered (lost or stolen) communications equipment.</td>
<td>- Secure equipment.</td>
</tr>
<tr>
<td></td>
<td>- Adversary is otherwise attempting to issue commands to platoon.</td>
<td>- Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares).</td>
</tr>
<tr>
<td><strong>E1A2. Platoon does not recognize Pl. because of communications equipment failure. H1,H3,H4,H5,H6.</strong></td>
<td>- Equipment not checked before mission.</td>
<td>- Check equipment prior to mission (functioning?, battery, antenna).</td>
</tr>
<tr>
<td></td>
<td>- Wrong equipment.</td>
<td>- Choose communication devices that will satisfy mission needs. Know operating distances, etc., of equipment.</td>
</tr>
<tr>
<td></td>
<td>- Battery dead, equipment fails in service.</td>
<td>- Bring back-up communication equipment, batteries, antenna.</td>
</tr>
<tr>
<td></td>
<td>- Equipment damaged or not functioning while on mission.</td>
<td>- Tools and know-how to troubleshoot and repair equipment.</td>
</tr>
<tr>
<td></td>
<td>- Lack of operational know how.</td>
<td>- Train soldiers on equipment use.</td>
</tr>
<tr>
<td></td>
<td>- Platoon not all on same channel.</td>
<td>- Communicate communication channel to subordinates.</td>
</tr>
<tr>
<td></td>
<td>- Terrain interference on transmission.</td>
<td>- Move to a better position.</td>
</tr>
<tr>
<td></td>
<td>- Communications equipment cannot otherwise be used.</td>
<td>- Contingency communications plan for equipment failure.</td>
</tr>
<tr>
<td><strong>E1A2. Platoon does not recognize Pl. for other reasons. H1,H3,H4,H5,H6.</strong></td>
<td>- PL incapacitated.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Change of leadership without sufficient transfer of command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Mutiny!</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- No line-of-sight for visual communications signals.</td>
<td></td>
</tr>
<tr>
<td><strong>E1A2. Platoon does not recognize Pl. for other reasons. H1,H3,H4,H5,H6.</strong></td>
<td>- Establish SOP for transfer of command. GOWA.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Establish SOP for transfer of command. GOWA.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Discourage mutiny! Foster confidence in leadership. Vet leaders.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Move within line-of-sight. Contingency communications for no line-of-sight. Animal calls, radios, etc.</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.18: Scenarios for UCA E1A2.
**E1A3**: state = any state, Authorized = No, Authenticated = Yes  
Table 6.19 present the scenarios for the E1A3 UCAs.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
</table>
| E1A3: PL indicates planning is complete when it is not. Platoon executes command because PL is authenticated. H1,H2,H4. | • Inadequate leadership.  
• Change of leadership without sufficient transfer of command.  
• Mutiny/rogue leadership.  
• Platoon does not recognize unauthorized requests. | • Vet leadership.  
• Establish SOP for transfer of command. GOTWA.  
• Discourage mutiny! Foster confidence in leadership. Vet leaders.  
• For the most part, soldiers follow orders. However, platoon should be trained on expectations regarding the operations and leader's duties and commands. Use common sense. |

Table 6.19: Scenarios for UCA E1A3.

**E1A4**: state = any state, Authorized = No, Authenticated = No  
Table 6.20 present the scenarios for the E1A4 UCAs.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Associated Causal Factors</th>
<th>Rationale</th>
</tr>
</thead>
</table>
| E1A4: Adversary issues command. H1,H3,H4,H5,H6. | • Adversary has commandeered (lost or stolen) communications equipment.  
• Adversary is otherwise attempting to issue commands to platoon. | • Secure equipment.  
• Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares). |

Table 6.20: Scenarios for UCA E1A4.
Table 6.21 describes scenarios for the *discard(planComplete)* control action.

<table>
<thead>
<tr>
<th>SCENARIOS</th>
<th>UCA: control action: discard</th>
</tr>
</thead>
<tbody>
<tr>
<td>state = any state, Authorized = Yes, Authenticated = Yes. Providing is an UCA. Discarding means that the platoon refuses to execute the command for some reason.</td>
<td><strong>Scenarios</strong></td>
</tr>
<tr>
<td>D1A1. Command discarded when it should be executed, H3,H4,H6,H8.</td>
<td></td>
</tr>
<tr>
<td>D1B1. Unauthenticated command not discarded. H3,H4,H6.</td>
<td></td>
</tr>
<tr>
<td>state = any state, Authorized = No, Authenticated = Yes. Providing is an UCA. Unauthorized commands are a problem with the Platoon Leader control structure...trickles down to the platoon. But, from the process model of the platoon, this command should not be discarded because it is authorized.</td>
<td></td>
</tr>
<tr>
<td>D1A2. Command discarded when it should be executed, H3,H4,H6,H8.</td>
<td></td>
</tr>
<tr>
<td>state = any state, Authorized = No, Authenticated = No. Not providing is a UCA.</td>
<td></td>
</tr>
<tr>
<td>D1B2. Unauthenticated command not discarded. H3,H4,H6</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.21: Scenarios for UCA for discard on all commands (D1A1 through D1A4).
6.5 Discussion And Conclusions

This chapter describes the STPA/STPA-Sec analysis on an abstract model of the patrol base operations. From this analysis, numerous recommendations regarding safety and security are derived (step 4). These recommendations ("Rationale" in the tables) would be passed to the systems engineers to inform the security aspects of the system design.

The reader who is familiar with the patrol base operations and military procedures in general may recognize the implementation of some of the rationale. For example, contingency plans are a standard for many phases of the operations. The Ranger Handbook explains basic repair techniques for communications equipment. METT-TC analysis is standard analysis for the patrol base operations. GOTWA is standard when a leader leaves the platoon. Some of the recommendation are common sense. For example, discourage mutiny and build confidence in leadership. Train soldiers to be disciplined. Secure equipment.

A more thorough analysis should find more recommendations that are specific to particular actions. For example, the troop leading procedures comprise an eight step planning phase. Each step could be analyzed using STPA/STPA-Sec to identify unsafe control actions that arise from system-level interactions.
Chapter 7

Patrol Base Operations

The patrol base operations are modeled as a modularized hierarchy of secure state machines to demonstrate CSBD on a non-automated, human-centered system. Once the operations are modeled as SSMs the parametrized SSMs in HOL can be used to verify that the model satisfies the principle of complete mediation.

This model is a collaborative effort between the author of this thesis and a subject matter expert Captain Jesse Nathaniel Hall from the U.S. Army. But, the details with regards to translating the Ranger Handbook into the model is the work of Captain Jesse Nathaniel Hall.

This chapter describes the model and the SSMs.

7.1 Patrol Base Operations

Patrol Base Operations: Mission Activity Specification

<table>
<thead>
<tr>
<th>Purpose</th>
<th>A system to establish a security perimeter when a squad or platoon halts for an extended period of time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>by means of planning, reconnaissance, security, control, and common sense</td>
</tr>
</tbody>
</table>
| Goal    | in order to • avoid detection  
|         | • hide a unit during a long, detailed reconnaissance  
|         | • perform maintenance on weapons, equipment, eat, and rest  
|         | • plan and issue orders  
|         | • reorganize after infiltrating an enemy area  
|         | • establish a base from which to execute several consecutive or concurrent operations |

Table 7.1: Mission Activity Specification for Patrol Base Operations. Adapted from the U.S. Army Ranger Handbook 2017 [9].

7.2 Overview of The Model

Each level of the hierarchy of secure state machine (SSM)s represents a level of abstraction of the patrol base operations. The most abstract level of the hierarchy is the top level SSM. A diagram of this most abstract level is shown in figure 7.1.

The diagram describes a chronological order of abstract phases (modeled as states) of the patrol base operations. (1) The operations begin with the planning phase. (2) Next, they move to the Objective Rally Point (ORP). (3) At the ORP, operations commence. (4) When these are complete, the patrol base operations move to the actual patrol base. (5) At the patrol base, operations proceed. (6) Finally, the patrol base operations are complete. The last phase is an end phase (actual withdraw, etc. of the operations are...
The next level of abstraction in the hierarchy of SSM represents a horizontal slice through the patrol base operations. This slice describes the patrol base operations at a lower level of abstraction. It expands each of the states in the top level (except for the last state PB Complete). For example, the planning phase (1) in figure 7.1 is expanded into an SSM of its own. This is called ssmPlanPB. It consists of several states (see section 7.3.6.1) which detail activities conducted during the planning phase of the patrol base operations.

At yet another lower level of abstraction is the sub-sub (3rd) level. This expands on the states in the level above it. This is the last level of the hierarchy that is modeled in detail.\(^1\)

To demonstrate complete mediation on all levels of the hierarchy, a vertical slice is also modeled. Each SSM expands upon only one state in the level above it, starting from the top level Move to ORP state and ending in a lower level state.

\(^1\)It is not necessary to model the entire system. To do so, is beyond the scope of this work
In addition to the horizontal and vertical slices, an escape level is also modeled. This level models actions that require the patrol base operations to abort. This is a floating module and can be reached from any state in the model.

### 7.3 Hierarchy of Secure State Machines

![Diagram of hierarchy of secure state machines](LaTeX/figures/diagram.vis)

Figure 7.2: Diagrammatic description of patrol base operations as a modularized hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)

#### 7.3.1 Diagrammatic Description in Visio

The enormity of the hierarchy of SSMs is evident in figure 7.2. This is a squashed version of the Visio diagram for the hierarchy of SSMs. The diagram is included as a Visio file with the files for this project (LaTeX/figures/diagram.vis).

**Levels of The Hierarchy** The straight, colored lines that span the diagram in figure 7.2 delineate levels of the hierarchy of SSM. The top lines are obscured by the size of this squashed version of the diagram. The most visible bright yellow line delineates the sub-sub-sub (4th) level of the hierarchy, for example.
Each level represents a different degree of abstraction with a set of principal responsible for actions at that level.

- **Red**—escape level:
  Unacceptable losses that require the mission to abort. This is a floating module that can be reached by any level.

- **Dark blue**—top level:
  Describes the patrol base operations as five phases (plus an end phase). The Platoon Leader (PL) is responsible at this level.

- **Blue grey**—sub-level:
  This level takes the previous level’s states to a lower level of abstraction. The Platoon Leader and Platoon Sergeant (PSG) are responsible for this level.

- **Grey green**—sub-sub-level:
  This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the patrol base headquarters: Platoon Leader, Platoon Sergeant, Radio Telephone Operator (RTO), Forward Observer (FO), Medic, and Heavy Weapons Squad Leader (HWSQL).

- **Bright yellow**—4th level:
  This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the squad leaders (SQL).

- **Orange**—5th level:
  This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the Team Leaders (TL).

- **Olive**—6th level:
  This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the Squads. That is, the squad behavior is described at this level and the squad indicates readiness.
• Dark green–7th level:
  This level takes the previous level’s states to a lower level of abstraction. The
  responsible principals are the Fire Team (FT), Recon Team (RT), Buddy Team
  (BT) and Security Team (ST).

• Beige–8th level:
  This level takes the previous level’s states to a lower level of abstraction. The
  responsible principals are the individual soldiers themselves.

The diagram is only a partial model of the patrol base operations. The horizontal slice
is the second to last row that expands across the diagram. The vertical slice is in the
middle of the diagram and expands several modules to the lowest level of abstraction.

**States and Transitions in The Hierarchy**  The small, colored dots in figure 7.2
represent states (phases) of the patrol base operations. The labels for these states are
not readable in this diagram because of limited space, but each dot contains a
description of the state. The dots are color coded, with a different color for dots (states)
at each level.

The lines connecting one dot to another represent transition requests. These lines are
annotated, but are also unreadable in this diagram.

The following sections describe modules from this diagram. The descriptions of the
states and transitions are discussed and a readable diagram for each model is provided.

### 7.3.2 Descriptions of Individual Modules

Each module is described diagrammatically in the following sections. They all follow a
similar pattern. The general pattern is discussed in this section. Exceptions are
discussed along with the diagrammatic descriptions for each individual module. (Note also that "module" and "SSM" are used interchangeably in the following sections.)

The advantage of modularity is that it simplifies managing complexity. It is easier to breakdown each state into a less abstract secure state machine than to model all possible transitions at one level. This also makes it easier to code in HOL (if doing it by hand). It also simplifies modifying the model. Specifically, making changes to one SSM does not require changes to any other SSM. To maintain this modularity, an entity named OMNI was used to communicate information from one module to another. In this way, information needed in one module could be communicated via OMNI without requiring the module using that information to have any information about how the module sending the information did this. While OMNI is an abstraction of the model, in the real system it would represent the principal communicating the information from one module to the other.

**Flow** Each module follows a sequential pattern. It starts at one state and then flows sequentially to the end of the module. Each module has a set of principals who are authorized on some set of transitions (or commands). Each module has its own security policy that dictates the conditions under which transition requests are granted.

**Requests And Security Policies** Principals make requests to transition from one state to another. Requests are of the form *Principal says command*.

There are two approaches to the policy. The first allows transition within each module with no regard to completion of lower level modules. For example, transition from the planning phase to the move to ORP phase in the top level does not require any specific information about completion of the planning module in the level below.

---

2The patrol base operations are modeled as sequential activities. However, not all operations follow a definite sequential pattern.
The second approach requires confirmation of completion of the lower level module before transitions at the top level are allowed. For example, before transitioning from the planning phase to the move to ORP phase, the planning phase must confirm that the planning module is complete. This isn’t necessarily true of the real patrol base operations. For example, the patrol may move to the ORP before receiving it’s mission. But, modeling it this way demonstrates integration modules at different levels of the hierarchy.

For this second approach, an "all knowing" principal (essentially a signal relay) communicates when a lower level module is complete. This principal is called OMNI. OMNI allows for encapsulation of each module by communicating information about one module to another and eliminating the need for principals in one module to "know" what is happening in another module. This is necessary to reason with the ACL in HOL. It is a feature of the model and not the operations.

**Diagrammatic Description** The colors of the states in the diagrams correspond to their colors in the overall compressed diagram shown in figure 7.2.

All lines represent allowable transitions with an arrow indicating the direction of the transition. Each line is annotated with the appropriate ACL request (or command). The last line in each module is an exception. It connects the COMPLETE state to the initial state. This line is not annotated. In SSMs that integrate modules, this could be thought of as feedback from OMNI.

**Naming conventions** What follows are the naming conventions for the diagrams. They also apply to the HOL implementation of the SSM.

**state:** all capital letters with underscores representing spaces. Examples include: MOVE_TO_ORP, PLAN_PB, etc.
commands (or requests): first letter is lower case. The remaining letters toggle with a capital letter for each new word. Examples include: moveToORP, receiveMission, etc. Furthermore, all commands take the name of the next state. For example, the transition from the state MOVE_TO_PB to CONDUCT_PB is conductPB. The transition from the state COMPLETE_PLAN to ISSUE_OPORD is issueOPORD. The only exception is the transition from the top level state PLAN_PB to the next state MOVE_TO_ORP. The command for this transition is crossLD and not moveToORP.

principals: all begin with a capital letter then follow the convention for commands (or requests). Examples include: PlatoonLeader, PlatoonSergeant, etc.

ACL transition requests: all are of the form Principal says command. Examples include: PlatoonLeader says moveToORP, PlatoonSergeant says actionsIn, etc.

7.3.3 OMNI-Level

OMNI is not represented in the Visio diagram. The main purpose of this entity is to relay messages from one SSM to another. This allows for greater encapsulation of the modules. To add OMNI to the diagram would over complicated and already complicated model.

In all SSM, OMNI is a principal who has authority over OMNI level commands. These commands communicate the completion of a lower-level SSM. For example, at the top level, before the Platoon Leader can transition from the PLAN_PB state to the MOVE_TO_ORP state, he must receive the command OMNI says ssmPlanPBComplete. The top level security policy contains the clause OMNI controls ssmPlanPBComplete. The Controls rule discussed in section 4.1 then allows the Platoon Leader to conclude that the lower-level SSM is complete.
7.3.4 Escape

A diagram of the escape level is shown in figure 7.3. The purpose of the escape level is to model situations wherein the patrol base operations must be aborted.

![Figure 7.3: Escape level diagram.](image)

The square boxes in the diagram represent communication from outside the module. They represent external signals from OMNI. The only state is the ABORT_MISSION state.

The abortable conditions are `returnToBase`, `changeMission`, `resupply`, and `reactToContact`.

The security policy for the escape level contains the clauses `OMNI controls returnToBase`, `OMNI controls changeMission`, etc. Given a request `OMNI says returnToBase` and the security policy, `returnToBase` is believed.

The security policy also contains a clause that allows the Platoon Leader to transition to the ABORT_MISSION state if `returnToBase (changeMission, etc.)` is believed and the Platoon Leader requests the transition.

---

3React to contact in this description refers to unwanted contact that requires the mission to be aborted. It does not refer to planned contact in combat missions or contact that does not require mission abortion.
7.3.5 Top Level

The top level for the hierarchy is shown in figure 7.4. This is a linearized version of figure 7.1.

![Top level diagram](image)

**Figure 7.4:** Top level diagram.

This SSM represents the phases shown in figure 7.1. There are six states: PLAN_PB, MOVE_TO_ORP, CONDUCT_ORP, MOVE_TO_PB, CONDUCT_PB, and COMPLETE_PB (an end state).

Commands to transition from one state to another are named after the next state. For example, to transition from MOVE_TO_ORP to CONDUCT_ORP, the command is conductORP. The transition from the PLAN_PB state to the MOVE_TO_ORP state is the only exception. This command is crossLD.

Transitions are initiated by a request from and authenticated and authorized principal. These requests have the form *PlatoonLeader says crossLD*.

This SSM integrates with the lower level modules. This means that transitions are allowed when the appropriate lower level module is complete. This requires the aid of the OMNI principal.

The Platoon Leader is the only authenticated principal at this level. Authentication is assumed to be visual or other recognition (i.e., does not require anything complicated such as a cryptographic key.)

The security policy is state dependent. It requires a signal from OMNI that the lower
level module is complete. Once this is believed, the Platoon Leader is authorized to make a transition to the next state.
7.3.6 Horizontal Slice

The horizontal slice is an expansion of the states in the top level SSM. Each state (save for the COMPLETE_PB state) is expanded into an SSM. These SSMs are described next.

7.3.6.1 ssmPlanPB

The top level PLAN_PB state is expanded into the ssmPlanPB SSM and shown in figure 7.5.

This is the largest SSM and models the eight steps of the troop leading procedures.

There are states: RECEIVE_MISSION, WARNO, TENTATIVE_PLAN, INITIATE_MOVEMENT, RECON, REPORT1, COMPLETE_PLAN, ISSUE_OPORD, SUPERVISE, REPORT2, and COMPLETE.

Transition commands follow the same pattern as in the other SSMs. The exception is discussed below.

All transitions are sequential. However, the original module contains three
non-sequential states. These are represented in the diagram as the white circle. These states are TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON. To transition from WARNO to REPORT1 requires that all three of these states be completed, but not in any specific order. To solve this problem, the three states are not represented as states. The completion of these "tasks" is indicated by the following three statements: PlatoonLeader says tentativePlan, PlatoonSergeant says initiateMovement, and PlatoonLeader says recon. Thus, the transition from WARNO to REPORT1 now requires four statements: PlatoonLeader says tentativePlan, PlatoonSergeant says initiateMovement, PlatoonLeader says recon, AND PlatoonLeader says report1. The latter-most statement is the actual request. The security policy enforces this transition by including the clause tentativePlan andf initiateMovement andf recon impf PlatoonLeader controls report1.

ssmPlanPB has two principals: PlatoonLeader and PlatoonSergeant. Only the PlatoonLeader is authorized to make transitions among states. The PlatoonSergeant controls the InitiateMovement command (but, it is not a state and therefore there is no transition).

This SSM does not integrate and thus does not require the OMNI principal. The security policy authorizes the Platoon Leader to make transitions as described in previous sections, with the exception as described above.
7.3.6.2 ssmMoveToORP

The top level MOVE_TO_ORP state is expanded into the ssmMoveToORP SSM and shown in figure 7.6.

![ssmMoveToORP SSM](image)

Figure 7.6: Horizontal slice: MoveToORP diagram.

This SSM has three states: FORM, MOVE, and SECURE_HALT.

The Platoon Leader is the only authorized principal.

This is an integrating SSM, and requires a signal from OMNI that the lower level SSM is complete.

The security policy allows for transitions if requested to do so from the Platoon Leader.
7.3.6.3 ssmConductORP

The top level CONDUCT.ORP state is expanded into the ssmConductORP SSM and shown in figure 7.7.

Figure 7.7: Horizontal slice: ConductORP diagram.

This SSM has four states: SECURE, ACTIONS_IN, WITHDRAW, and COMPLETE.

There are two principals. The Platoon Leader is authorized on all commands except for the actionsIn command. The PlatoonSergeant is only authorized on the actionsIn command.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader or the PlatoonSergeant.
7.3.6.4 ssmMoveToPB

The top level MOVE_TO_PB state is expanded into the ssmMoveToPB SSM and shown in figure 7.8.

This SSM is the same as ssmMoveToORP. It has three states: FORM, MOVE, and SECURE_HALT.

The Platoon Leader is the only authorized principal.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.
The top level CONDUCT_PB state is expanded into the ssmConductPB SSM and shown in figure 7.9.

![Horizontal slice: ConductPB diagram.]

This SSM is the same as ssmConductORP. It has four states: SECURE, ACTIONS_IN, WITHDRAW, and COMPLETE.

There are two principals. The Platoon Leader is authorized on all commands except for the actionsIn command. The PlatoonSergeant is only authorized on the actionsIn command.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader or the PlatoonSergeant.
7.3.7 Vertical Slice

The vertical slice is an expansion of one state at each level of the hierarchy. It is the middle section in the overall, squished diagram in figure 7.2. This is the only section of the patrol base operations that are modeled through to the lowest level of abstraction.

The vertical slice starts at the top level state MOVE_TO_ORP. This state is expanded into the ssmMoveToORP SSM. In this sub level, the state SECURE_HALT is expanded into the ssmSecureHalt SSM. In this sub-sub-level SSM, the state ORP_RECON is expanded into the ssmORPRecon SSM. In this SSM, the state MOVE_TO_ORP is expanded into the ssmMoveToORP4L SSM. In this SSM, the state FORM_RT is expanded into the ssmFormRT SSM.
7.3.7.1  ssmSecureHalt

The sub-sub-level SECURE_HALIT state is expanded into the ssmSecureHalt SSM and shown in figure 7.10.

This SSM has four states: SECURE, ORP_RECON, WITHDRAW, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.
7.3.7.2 ssmORPRecon

The sub-sub-sub-level ORP_RECON state is expanded into the ssmORPRecon SSM and shown in figure 7.11.

![Diagram of ssmORPRecon SSM](image)

**Figure 7.11:** Vertical slice: ORPRecon diagram.

This SSM has six states: CONTINGENCY_PLAN, MOVE_TO_ORP, CONDUCT_ORP, FORM_ST, RETURN_TO_UNIT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.
7.3.7.3  ssmMoveToORP4L

The 4th level MOVE_TO_ORP state is expanded into the ssmMoveToORP4L SSM and shown in figure 7.12. Recall that there is no module at the 5th level.

![Figure 7.12: Vertical slice: MoveToORP4L diagram.](image-url)

This SSM has four states: FORM_RT, RT_MOVE, RT_HALT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.
7.3.7.4 ssmFormRT

The 6th level FORM_RT state is expanded into the ssmFormRT SSM and shown in figure 7.12. (Note that there is no module at the 5th level for this slice. Also, the 8th level is not modeled.)

Figure 7.13: Vertical slice: FormRT diagram.

This SSM has four states: RT_POSITION, RT_ORIENT, RT_ALERT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

7.4 Discussion

The modularized hierarchy of SSMs eased the analysis process. This lead us to think about the operations in a non-standard way. If forced us to look at the command and
control structure, which is the goal of CSBD. For each action, we determined who was responsible either implicitly or explicitly.

The hierarchy also demonstrated the chain of command. For example, responsibility at the top level was delegated to the platoon leader who is ultimately responsible for completing the mission and bring his troops home safely.

The areas where we found difficulties were in maintaining modularity when actions at one level affected actions at another level in a non-modularizable way. But the basic solution was the OMNI-level principal who had authority in all SSMs.

The next chapter describes how a representative selection of the SSMs described in this chapter are implemented in HOL.
Chapter 8

 Patrol Base Operations as Secure State Machines

This chapter details a representative subset of the SSMs described in the previous chapter. The code for all of the SSMs can be found in the appendices.

Many of the SSMs are similar and differ only in the names of the states, commands, and principals. The first SSM, ssmPB, is a typical example from the hierarchy that uses the OMNI principal and one other principal. All transitions are sequential. The next example, ssmConductORP, uses two principals and the OMNI principal. The third example is ssmPlanPB, which contains the non-sequential conjunction of states. It does not use the OMNI principal. All other SSMs fit into one of these three patterns.
8.1 ssmPB: A Typical Example from the Hierarchy

ssmPB is the top level SSM. It is an example of a typical one-principal\(^1\) SSM that also uses the Omni principal. A diagram of ssmPB is shown in figure 8.1.

![Figure 8.1: Top level diagram.](image)

ssmPB runs sequentially from the initial state PLAN_PB to the final state COMPLETE_PB. State transitions require notification from the Omni principal that the state is complete and a request from the Platoon Leader to change states. Thus, a transition request is a list and has the form\(^2\)

\[
\begin{align*}
\text{Omni says stateComplete;} \\
\text{PlatoonLeader says moveToNextState}
\end{align*}
\]

The security context reflects this two input requirement. It authorizes the Platoon Leader on \textit{moveToTheNextState} if the current state is complete. A clause in the security context has the form\(^3\)

\[
\text{stateComplete impf}
\]

\[
\text{PlatoonLeader controls moveToNextState}
\]

\(^1\)For the purposes of transitions, the Omni principal isn’t thought of as a principal, but a relay. However, for the purposes of defining principals in HOL, Omni is included as a principal.

\(^2\)stateComplete and moveToNextState are place holders for the actual commands which are defined later.

\(^3\)impf is the ACL-HOL equivalent of "implies"
This clause is state dependent because `stateComplete` is specific for the current state. Thus, there are five statements in the state-dependent security context because there are six states.

There is also a state-independent security context. This has one clause authorizing the Omni principal on all omniCommands. It has the form

\[ Omni \text{ controls } stateComplete \]

The HOL implementation of the datatypes, functions, and theorems are included below.

### 8.1.1 Principals

The principal datatype is defined in OMNITypeScript.sml. It has a constructor and one datatype variable.

\[ principal = SR \quad \text{'stateRole} \]

\( SR \) is the type constructor and \( \text{'stateRole} \) is the type variable. Each SSM defines its own principal as a \( \text{stateRole} \). In ssmPB, the \( \text{stateRole} \) datatype has two principals.

\[ stateRole = \text{PlatoonLeader} \mid \text{Omni} \]

### 8.1.2 States

States are also defined in OMNITypeScript.sml.

\[ state = \text{ESCs escState} \mid \text{SLs 'slState} \]
'slState is a state-dependent state which is further defined in each SSM. ssmPB defines six states.

\[ slState = \text{PLAN\_PB} \]
\[ \quad | \text{MOVE\_TO\_ORP} \]
\[ \quad | \text{CONDUCT\_ORP} \]
\[ \quad | \text{MOVE\_TO\_PB} \]
\[ \quad | \text{CONDUCT\_PB} \]
\[ \quad | \text{COMPLETE\_PB} \]

8.1.3 Outputs

Outputs are defined in OMNITyrpeScript.sml.

\[ output = \text{ESCo escOutput} | \text{SLo 'slOutput} \]

SLo 'slOutput is the state-dependent output. It is defined further in each SSM. ssmPB defines seven outputs.

\[ slOutput = \text{PlanPB} \]
\[ \quad | \text{MoveToORP} \]
\[ \quad | \text{ConductORP} \]
\[ \quad | \text{MoveToPB} \]
\[ \quad | \text{ConductPB} \]
\[ \quad | \text{CompletePB} \]
\[ \quad | \text{unAuthenticated} \]
\[ \quad | \text{unAuthorized} \]

unAuthorized and unAuthenticated pertain to trap and discard transition types,
respectively.

## 8.1.4 Commands

OMNITyrpeScript.sml defines datatypes for all the SSMs.

\[\text{command} = \text{ESCC escCommand} | \text{SLc 'slCommand}\]

\text{SLc 'slCommand} is further defined in each SSM. ssmPB defines two datatypes for the \text{slCommand}.

\[\text{slCommand} = \text{PL plCommand} | \text{OMNI omniCommand}\]

The \text{omniCommand} and \text{plCommand} are further defined in ssmPB.

\[\text{omniCommand} = \text{ssmPlanPBComplete}\]
\[| \text{ssmMoveToORPComplete}\]
\[| \text{ssmConductORPComplete}\]
\[| \text{ssmMoveToPBComplete}\]
\[| \text{ssmConductPBComplete}\]
\[| \text{invalidOmniCommand}\]

\[\text{plCommand} = \text{crossLD}\]
\[| \text{conductORP}\]
\[| \text{moveToPB}\]
\[| \text{conductPB}\]
\[| \text{completePB}\]
\[| \text{incomplete}\]

101
8.1.5 Next-State Function

Each SSM defines its own next-state function. This is the formalization of the
next-state diagram shown in figure 8.1.

[PBNS_def]

\[
(PBNS \text{ PLAN\_PB} (\text{exec } x) =
\quad \text{if } \text{getPlCom } x = \text{crossLD then MOVE\_TO\_ORP else PLAN\_PB}) \land
\quad (PBNS \text{ MOVE\_TO\_ORP} (\text{exec } x) =
\quad \quad \text{if } \text{getPlCom } x = \text{conductORP then CONDUCT\_ORP}
\quad \quad \quad \text{else MOVE\_TO\_ORP}) \land
\quad (PBNS \text{ CONDUCT\_ORP} (\text{exec } x) =
\quad \quad \text{if } \text{getPlCom } x = \text{moveToPB then MOVE\_TO\_PB else CONDUCT\_ORP}) \land
\quad (PBNS \text{ MOVE\_TO\_PB} (\text{exec } x) =
\quad \quad \text{if } \text{getPlCom } x = \text{conductPB then CONDUCT\_PB else MOVE\_TO\_PB}) \land
\quad (PBNS \text{ CONDUCT\_PB} (\text{exec } x) =
\quad \quad \text{if } \text{getPlCom } x = \text{completePB then COMPLETE\_PB}
\quad \quad \quad \text{else CONDUCT\_PB}) \land (PBNS \ s (\text{trap } v_0) = s) \land
\quad (PBNS \ s (\text{discard } v_1) = s)
\]

PBNS takes a state and a transition \((exec, \ trap, \ or \ discard)\) command list \((x)\) as
parameters.

The next-state function for ssmPB uses pattern matching and if-then-else statements.
The function first matches the state. Then, it uses the \text{getPlCom} function to extract the
\text{plCommand} from the command list \(x\). It compares this command to the command on
the right. If they are equal, then the transition occurs. Otherwise, the state does not
change.

For the \text{trap} and \text{discard} transition types (last two lines in the definition), no transition
occurs regardless of the command. Therefore, the command is not checked. The original state $s$ is returned.

8.1.6 Next-Output Function

Each SSM defines its own next-state function.

[PBOut_def]

\[
\triangleright (\text{PBOut PLAN_PB (exec } x) = \\
\quad \text{if } \text{getPlCom } x = \text{crossLD then MoveToORP else PlanPB) } \land \\
(\text{PBOut MOVE_TO_ORP (exec } x) = \\
\quad \text{if } \text{getPlCom } x = \text{conductORP then ConductORP else MoveToORP) } \land \\
(\text{PBOut CONDUCT_ORP (exec } x) = \\
\quad \text{if } \text{getPlCom } x = \text{moveToPB then MoveToORP else ConductORP) } \land \\
(\text{PBOut MOVE_TO_PB (exec } x) = \\
\quad \text{if } \text{getPlCom } x = \text{conductPB then ConductPB else MoveToPB) } \land \\
(\text{PBOut CONDUCT_PB (exec } x) = \\
\quad \text{if } \text{getPlCom } x = \text{completePB then CompletePB else ConductPB) } \land \\
(\text{PBOut } s \ (\text{trap } v_0) = \text{unAuthorized}) \land \\
(\text{PBOut } s \ (\text{discard } v_1) = \text{unAuthenticated})
\]

The next-output function behaves similarly to the next-state function. But, instead of returning the next state, it returns the next output. The trap and discard transition types return unAuthorized and unAuthenticated, respectively, for any command list $x$.  

103
8.1.7 Authentication

The *authenticationTest* function is defined in the parametrizable ssm. It takes a function as an input and that function takes an input list as an input. The first function is named *elementTest* in ssm. It is named *inputOK* in the SSMs.

In HOL, *inputOK* uses the wild card denoted by an underscore "_". This causes HOL to generate a lot of additional code when it evaluates the definition. Both the HOL definition and the HOL-generated output for this function are shown below.

val inputOK_def =

Define

\[
\begin{align*}
\text{inputOK} & \quad \text{(((Name PlatoonLeader) says prop (cmd:\text{(slCommand command)option}))} \\
& \quad :\text{(slCommand command)} \text{option , stateRole , 'd, 'eForm} = \text{T}) \quad /\ \\
\text{inputOK} & \quad \text{(((Name Omni) says prop (cmd:\text{(slCommand command)option}))} \\
& \quad :\text{(slCommand command)} \text{option , stateRole , 'd, 'eForm} = \text{T}) \quad /\ \\
\text{inputOK} & \quad _ = \text{F})
\end{align*}
\]

*inputOK* authenticates only the Platoon Leader and Omni on state-level commands (slCommands) and on no other commands. No other principals are authenticated for any command.

The HOL definition for *inputOK* uses pattern matching. The first call to *inputOK* matches the input to

\[
\text{(((Name PlatoonLeader) says prop (cmd:\text{(slCommand command)option}))}.
\]

If it matches, the function returns T for true. The second call to *inputOK* matches the input to

\[
\text{(((Name Omni) says prop (cmd:\text{(slCommand command)option}))}.
\]

If it matches, it returns true. The last call to *inputOK* uses the wild card. This returns false for any other input.

---

4The function *elementTest* should be thought of as a parameter where *inputOK* specializes that parameter.
The first two calls to `inputOK` authenticate the PlatoonLeader and Omni principals on any `slCommand`.

HOL Generated Output for `inputOK`  

```

\[
\begin{align*}
\vdash (inputOK (Name \text{PlatoonLeader} \text{ says } \text{prop cmd}) \iff T) \land \\
& (inputOK (Name \text{Omni} \text{ says } \text{prop cmd}) \iff T) \land \\
& (inputOK \text{TT} \iff F) \land (inputOK \text{FF} \iff F) \land \\
& (inputOK (\text{prop v}) \iff F) \land (inputOK (\text{notf v}_1) \iff F) \land \\
& (inputOK (v_2 \text{ andf v}_3) \iff F) \land (inputOK (v_4 \text{ orf v}_5) \iff F) \land \\
& (inputOK (v_6 \text{ impf v}_7) \iff F) \land (inputOK (v_8 \text{ eqf v}_9) \iff F) \land \\
& (inputOK (v_{10} \text{ says TT}) \iff F) \land (inputOK (v_{10} \text{ says FF}) \iff F) \land \\
& (inputOK (v_{133} \text{ meet v}_{134} \text{ says } \text{prop v}_{66}) \iff F) \land \\
& (inputOK (v_{135} \text{ quoting v}_{136} \text{ says } \text{prop v}_{66}) \iff F) \land \\
& (inputOK (v_{10} \text{ says notf v}_{67}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } (v_{68} \text{ andf v}_{69})) \iff F) \land \\
& (inputOK (v_{10} \text{ says } (v_{70} \text{ orf v}_{71})) \iff F) \land \\
& (inputOK (v_{10} \text{ says } (v_{72} \text{ impf v}_{73})) \iff F) \land \\
& (inputOK (v_{10} \text{ says } (v_{74} \text{ eqf v}_{75})) \iff F) \land \\
& (inputOK (v_{10} \text{ says } (v_{76} \text{ says v}_{77})) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{78} \text{ speaks_for v}_{79}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{80} \text{ controls v}_{81}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } \text{reps v}_{82} v_{83} v_{84}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{85} \text{ domi v}_{86}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{87} \text{ eqi v}_{88}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{89} \text{ doms v}_{90}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{91} \text{ eqs v}_{92}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{93} \text{ eqn v}_{94}) \iff F) \land \\
& (inputOK (v_{10} \text{ says } v_{95} \text{ lte v}_{96}) \iff F) \land
\end{align*}
\]
```

105
It is straightforward to prove that any command that is not issued by a principal is rejected. This follows directly from the definition of \texttt{inputOK}.

\texttt{[inputOK\_cmd\_reject\_lemma]}

\[\vdash \forall \textit{cmd}. \neg \text{inputOK } (\text{prop } (\text{SOME cmd}))\]

\subsection{8.1.8 Authorization}

There are two functions for authorization in the parametrizable ssm. One is state-dependent, the other is not.

**State-dependent Authorization**  In ssmPB, the state-dependent authorization function is named \texttt{secContext}. \texttt{secContext} uses both pattern matching and if-then-else statements. It takes a state and an input list as parameters.

\texttt{[secContext\_def]}

\[\vdash (\forall \textit{xs} \cdot \hspace{1em} \texttt{secContext PLAN_PB xs} = \]

106
if getOmniCommand xs = ssmPlanPBComplete then
    [prop (SOME (SLc (OMNI ssmPlanPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL crossLD)))]
else [prop NONE]) ∧

(∀xs.
secContext MOVE_TO_ORP xs =
    if getOmniCommand xs = ssmMoveToORPComplete then
        [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
         Name PlatoonLeader controls
         prop (SOME (SLc (PL conductORP)))]
    else [prop NONE]) ∧

(∀xs.
secContext CONDUCT_ORP xs =
    if getOmniCommand xs = ssmConductORPComplete then
        [prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
         Name PlatoonLeader controls
         prop (SOME (SLc (PL moveToPB)))]
    else [prop NONE]) ∧

∀xs.
secContext MOVE_TO_PB xs =
    if getOmniCommand xs = ssmConductORPComplete then
        [prop (SOME (SLc (OMNI ssmMoveToPBComplete))) impf
         Name PlatoonLeader controls
         prop (SOME (SLc (PL conductPB)))]
    else [prop NONE]) ∧

∀xs.
secContext CONDUCT_PB xs =
    if getOmniCommand xs = ssmConductPBComplete then
        [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
         Name PlatoonLeader controls
Running through an example, if the patrol base operations are in the PLAN_PB state and if the Platoon Leader says ssmPlanPBComplete, then secContext concludes that the Platoon Leader controls crossLD. If, however, the patrol base operations are in the PLAN_PB state and the Platoon Leader does not say ssmPlanPBComplete, then secContext returns NONE. The same logic follows for the remaining states.

secContext uses a helper function named getOmniCommand to extract the omniCommand from the input list. It compares this command to the expected command. If it matches, it returns an implication (impf) of the form described in section 8.1.

**HOL Definition for getOmniCommand**

getOmniCommand also uses the wildcard "_". For brevity, only the HOL definition is presented.

```
val getOmniCommand_def =
Define 'getOmniCommand ([]:((slCommand command)option , stateRole , 'd,'e)Form list )
 = invalidOmniCommand:omniCommand) /
 (getOmniCommand (((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs))
 = (cmd:omniCommand)) /
 (getOmniCommand ((x:((slCommand command)option , stateRole , 'd,'e)Form)::xs)
 = (getOmniCommand xs))`
```

getOmniCommand uses pattern matching and recursion. The first pattern matches the input against the empty list. This returns invalidOmniCommand. The second pattern matches against a statement of the form Omni says cmd. It returns cmd. The final pattern is a recursive call to getOmniCommand. Thus, if no valid command is present then invalidOmniCommand is returned, otherwise the valid cmd is returned.
State-independent Authorization  The state-independent authorization function
\textit{secAuthorization} takes an input list as a parameter. It uses a helper function called
\textit{secHelper}. \textit{secHelper} calls \textit{getOmniCommand}. It returns a authorization statement of
the form

\textit{Omni controls cmd}.

[secAuthorization\_def]

\[\forall xs. \text{secAuthorization } xs = \text{secHelper } (\text{getOmniCommand } xs)\]

[secHelper\_def]

\[\forall cmd. \]

\[\text{secHelper } cmd = \]

\[[\text{Name Omni controls prop }\ (\text{SOME } (\text{SLc } (\text{OMNI cmd})))]\]

8.1.9 Proved Theorems

These theorems with their helper lemmas prove the property of complete mediation for
each transition in the SSM.

PlatoonLeader\_PLAN\_PB\_exec\_justified\_thm  This theorem proves that
transition from the PLAN\_PB state to the MOVE\_TO\_ORP state is justified for the
following assumptions:

- The current state is PLAN\_PB
- Omni says ssmPlanPBComplete
PlatoonLeader says crossLD

Most proofs are similar to this. Therefore, this first example is described in detail and should be used as a reference for other proofs.

\textit{PlatoonLeader\_PLAN\_PB\_exec\_justified\_thm} begins by specializing \textit{TR\_exec\_cmd\_rule} described in chapter 5. That rule is repeated here for reference.

\texttt{[TR\_exec\_cmd\_rule]}

\[
\forall \text{elementTest context stateInterp } x \text{ ins } s \text{ outs}.
\]
\[
(\forall M \ Oi \ Os.
\text{CFGInterpret } (M, Oi, Os)
\quad (\text{CFG elementTest stateInterp context } (x::ins) \ s
\quad outs) \Rightarrow
\quad (M, Oi, Os) \text{ satList propCommandList } x) \Rightarrow
\quad \forall NS \ Out M \ Oi \ Os.
\]
\[
\text{TR } (M, Oi, Os) \text{ (exec (inputList } x))
\quad (\text{CFG elementTest stateInterp context } (x::ins) \ s \ outs)
\quad (\text{CFG elementTest stateInterp context } ins
\quad (NS \ s \text{ (exec (inputList } x)))
\quad (Out \ s \text{ (exec (inputList } x))::outs)) \iff
\quad \text{authenticationTest elementTest } x \land
\quad \text{CFGInterpret } (M, Oi, Os)
\quad (\text{CFG elementTest stateInterp context } (x::ins) \ s \ outs) \land
\quad (M, Oi, Os) \text{ satList propCommandList } x
\]

Specializing requires use of the ISPECL rule. This rule takes a list of parameters (lists are enclosed in square brackets) and a theorem (the theorem to be specialized).

\texttt{val thPlanPB = ISPECL}
thPlanPB is a temporary function used to store an intermediate value. In it, the following values are substituted into the TR_exec_cmd_rule.

- `inputOK`: substitutes for `elementTest`
- `secAuthorization`: substitutes for `context`
- `secContext`: substitutes for `stateInterp`
- an input list: substitutes for `s`
- the current state: substitutes for `x`
- an output list: substitutes for `outs`

The TR_exec_cmd_rule is already proved in the parametrizable ssm. It is an implication with a hypothesis and a conclusion. But, with the specialization, it is necessary to prove that the theorem is valid after the authentication and authorization are applied to the input list. To do this, the thPlanPB is deconstructed into a hypothesis and conclusion. The hypothesis and conclusion are proved as lemmas. The lemmas are then used to prove the overall theorem.
The first lemma begins with the hypothesis of \textit{thPlanPB}. It extracts the hypothesis using the \texttt{dest_imp} to destroy the implication and the \texttt{fst} function to retain the first part of the result.

\[
\text{fst(dest_imp(concl thPlanPB)))}
\]

The result is a HOL term which is used in a tactical proof (backwards proof).

\[
\text{val PlatoonLeader\_PLAN\_PB\_exec\_lemma =}
\]

\[
\text{TAC\_PROOF(}
\]

\[
\text{([],fst(dest_imp(concl thPlanPB))),}
\]

\[
\text{REWRITE\_TAC[CFGInterpret\_def,secContext\_def,secAuthorization\_def,secHelper\_def,}
\]

\[
\text{propCommandList\_def,extractPropCommand\_def,inputList\_def,}
\]

\[
\text{getOmniCommand\_def,}
\]

\[
\text{MAP,extractInput\_def,satList\_CONS,satList\_nil,GSYM satList\_conj] THEN}
\]

\[
\text{PROVE\_TAC[Controls,Modus\_Ponens]})}
\]

The proof consists of a call to the \texttt{REWRITE\_TAC} with several definitions passed as parameters (in brackets). \texttt{REWRITE\_TAC}\textsuperscript{5} rewrites the lemma using the appropriate definition shown in brackets. The definitions in brackets are defined earlier in this and other chapters. The last line of the proof uses \texttt{PROVE\_TAC}, an automatic prover, with the Controls and Modus\_Ponens rules. These rules are described in the HOL representation of ACL in chapter 4.

This lemma is saved as \textit{PlatoonLeader\_PLAN\_PB\_exec\_lemma}. The HOL-generated output is shown below.

\[
\text{[PlatoonLeader\_PLAN\_PB\_exec\_lemma]}
\]

\[
\vdash \forall M\ Oi\ Os.
\]

\[
\text{CFGInterpret (M,Oi,Os)}
\]

\[
\text{(CFG inputOK secContext secAuthorization}
\]

\[
\text{([Name Omni says}
\]

\[
\text{prop (SOME (SLc (OMNI ssmPlanPBComplete)))]);}
\]

\textsuperscript{5}See https://hol-theorem-prover.org/kananaskis-11-helpdocs/help/HOLindex.html for details on HOL tactics and functions.
The next lemma works on the conclusion of \textit{thPlanPB}. It extracts the conclusion using the \textit{dest\_imp} and \textit{snd} rules.

It is also a tactical proof that uses PROVE\_TAC. PROVE\_TAC takes two parameters. The first is the previous lemma and the second is \textit{TR\_exec\_cmd\_rule}.

This lemma is saved as \textit{PlatoonLeader\_PLAN\_PB\_exec\_justified\_lemma}. The HOL-generated output is shown below.
prop (SOME (SLc (PL crossLD)))))
(CFG inputOK secContext secAuthorization
([Name Omni says
    prop (SOME (SLc (OMNI ssmPlanPBComplete))));
Name PlatoonLeader says
    prop (SOME (SLc (PL crossLD))):::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(exec
    (inputList
        [Name Omni says
            prop (SOME (SLc (OMNI ssmPlanPBComplete))));
        Name PlatoonLeader says
            prop (SOME (SLc (PL crossLD)))])))

(Out PLAN_PB
(exec
    (inputList
        [Name Omni says
            prop (SOME (SLc (OMNI ssmPlanPBComplete))));
        Name PlatoonLeader says
            prop (SOME (SLc (PL crossLD))))::outs)) ↔
authenticationTest inputOK
    [Name Omni says
        prop (SOME (SLc (OMNI ssmPlanPBComplete))));
    Name PlatoonLeader says
        prop (SOME (SLc (PL crossLD)))] ∧
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
    prop (SOME (SLc (OMNI ssmPlanPBComplete))));
Name PlatoonLeader says
    prop (SOME (SLc (PL crossLD)))]])
prop \((\text{SOME (SLc (PL crossLD)))})::\text{ins}\) PLAN_PB

\( \text{outs} \) \&

\((M, Oi, Os) \text{ satList}

propCommandList

[Name Omni says

prop \((\text{SOME (SLc (OMNI ssmPlanPBComplete)))})\];

Name PlatoonLeader says prop \((\text{SOME (SLc (PL crossLD)))})\]

With this last lemma, the main theorem is proved using a simple forward proof. The proof uses the REWRITE_RULE with several theorems including the previous lemma.

\texttt{val PlatoonLeader\_PLAN\_PB\_exec\_justified\_thm = REWRITE\_RULE[\text{inputList\_def}, \text{extractInput\_def}, \text{MAP}, \text{propCommandList\_def},
\text{extractPropCommand\_def}, \text{PlatoonLeader\_PLAN\_PB\_exec\_lemma}]

PlatoonLeader\_PLAN\_PB\_exec\_justified\_lemma}

The HOL-generated output is shown below.

\texttt{[PlatoonLeader\_PLAN\_PB\_exec\_justified\_thm]}

\(\forall NS Out M Oi Os.\)

\(\text{TR (M, }Oi, Os)\)

\((\text{exec}
\begin{align*}
\text{[SOME (SLc (OMNI ssmPlanPBComplete))];
\text{SOME (SLc (PL crossLD)))]}
\end{align*}

\text{CFG inputOK secContext secAuthorization}

\{\text{Name Omni says

prop \((\text{SOME (SLc (OMNI ssmPlanPBComplete)))})\};

Name PlatoonLeader says

prop \((\text{SOME (SLc (PL crossLD)))})::\text{ins}\) PLAN_PB \text{ outs})

\text{CFG inputOK secContext secAuthorization ins}

\(\text{NS PLAN_PB}
\begin{align*}
\text{(exec}
\text{[SOME (SLc (OMNI ssmPlanPBComplete))];
\end{align*}

115
This theorem demonstrates the logical soundness of the CONOPs for this mission in this state, namely that transition from the PLAN_PB state to the MOVE_TO_ORP state is executed if and only if the command was both authentication and authorized. In other words, this theorem proves that if the command was executed then it was authenticated and authorized and vis-a-vis.

With few exceptions, all proofs justifying execution of a command in the SSMs follow the same format.
PlatoonLeader_PLAN_PB_trap_justified_thm

PlatoonLeader_PLAN_PB_trap_justified_thm proves the trap transition is justified in the following case

- current state is PLAN_PB
- Omni says someOmniCommand and someOmniCommand ≠ ssmPlanPBComplete
- PlatoonLeader says crossLD

This is a trap because both Omni and PlatoonLeader are authenticated. But, PlatoonLeader is not authorized to transition to the next state unless Omni says ssmPlanPBComplete.

PlatoonLeader_PLAN_PB_trap_justified_thm follows the same pattern of implications destruction and lemma proofs. The difference is that the TR_trap_cmd_rule is specialized instead of the TR_exec_cmd_rule. The two lemmas and theorem are shown below as pretty-printed HOL-generated output.

[PlatoonLeader_PLAN_PB_trap_lemma]

\[\text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow \]
\[ (s = \text{PLAN_PB}) \Rightarrow \]
\[ \forall M\ Oi\ Os.\]
\[ \text{CFGInterpret} (M, Oi, Os) \]
\[ (\text{CFG inputOK secContext secAuthorization} \]
\[ (\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))));} \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL crossLD))))::ins} \text{ PLAN_PB} \]
\[ outs) \Rightarrow \]
\[ (M, Oi, Os) \text{ sat prop NONE} \]
\[ \text{PlatoonLeader}_\text{PLAN_PB_trap_justified_lemma} \]

\[ \vdash \text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow \\
( s = \text{PLAN_PB} ) \Rightarrow \\
\forall NS \ Out M Oi Os. \\
\text{TR} ( M, Oi, Os ) \\
\text{(trap)} \\
\text{(inputList)} \\
\text{[Name Omni says prop (SOME (SLc (OMNI omniCommand)))]}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))}}) \\
\text{(CFG inputOK secContext secAuthorization)} \\
\text{(([Name Omni says prop (SOME (SLc (OMNI omniCommand)))]);} \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))::ins) PLAN_PB outs)} \\
\text{(CFG inputOK secContext secAuthorization ins)} \\
\text{(NS PLAN_PB)} \\
\text{(trap)} \\
\text{(inputList)} \\
\text{[Name Omni says prop (SOME (SLc (OMNI omniCommand)))]}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))}}) \\
\text{(Out PLAN_PB)} \\
\text{(trap)} \\
\text{(inputList)} \\
\text{[Name Omni says prop (SOME (SLc (OMNI omniCommand)))]}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))}}) \\
\text{::outs}) \iff \]

118
authenticationTest inputOK

[Name Omni says prop (SOME (Slc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (Slc (PL crossLD)))] \land

CFGInterpret \((M, O_i, O_s)\)

\((\text{CFG inputOK secContext secAuthorization}

\quad([\text{Name Omni says prop (SOME (Slc (OMNI omniCommand)))};
\quad\text{Name PlatoonLeader says}
\quad\text{prop (SOME (Slc (PL crossLD)))]}::\text{ins}) \text{ PLAN_PB outs}) \land (M, O_i, O_s) \text{ sat prop NONE}

\[\text{PlatoonLeader\_PLAN\_PB\_trap\_justified\_thm}]\)

\[\vdash \text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow
\quad(s = \text{PLAN_PB}) \Rightarrow
\quad\forall NS \ Out M O_i O_s.
\quad\text{TR} (M, O_i, O_s)
\quad(\text{trap}
\quad\quad[SOME (Slc (OMNI omniCommand));
\quad\quad\text{SOME (Slc (PL crossLD))})]
\quad\quad(\text{CFG inputOK secContext secAuthorization}
\quad\quad\quad([\text{Name Omni says prop (SOME (Slc (OMNI omniCommand)))};
\quad\quad\quad\text{Name PlatoonLeader says}
\quad\quad\quad\text{prop (SOME (Slc (PL crossLD)))]}::\text{ins}) \text{ PLAN_PB outs})
\quad\quad(\text{CFG inputOK secContext secAuthorization} \ ins
\quad\quad\quad(\text{NS} \text{ PLAN_PB}
\quad\quad\quad(\text{trap}
\quad\quad\quad\quad[SOME (Slc (OMNI omniCommand));
\quad\quad\quad\quad\text{SOME (Slc (PL crossLD))})])
\quad\quad\quad(\text{Out} \text{ PLAN_PB}
\quad\quad\quad(\text{trap

119
PlatoonLeader_Omni_notDiscard_slCommand_thm  This theorem proves that if the PlatoonLeader issues an omniCommand and Omni issues a plCommand, the command is not discarded. The reason for this is that plCommand and omniCommand are both slCommands and inputOK authenticates both principals for slCommands. What this means more generally is that both the PlatoonLeader and Omni are authenticated on any state-level command.

This proof specializes the TR_discard_cmd_rule. This follows a different pattern with no lemmas. The HOL-generated output is shown below.

\[\forall NS\ Out\ M\ Oi\ Os.\]

\[\neg TR\ (M, Oi, Os)\]

\([\text{discard}\]

\[\begin{array}{l}
  \text{[SOME (SLc (OMNI omniCommand));} \\
  \text{SOME (SLc (PL crossLD)))]}::\text{outs})\end{array}\]

\[\implies\]

authenticationTest inputOK

\[\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});\]

\[\text{Name PlatoonLeader says} \]

\[\text{prop (SOME (SLc (PL crossLD)))} \wedge\]

\[\text{CFGInterpret} \ (M, Oi, Os)\]

\[\text{(CFG inputOK secContext secAuthorization}\]

\[\begin{array}{l}
  \text{([Name Omni says prop (SOME (SLc (OMNI omniCommand))});} \\
  \text{Name PlatoonLeader says} \\
  \text{prop (SOME (SLc (PL crossLD)))]}::\text{ins)) PLAN_PB} \\
  \text{outs}) \wedge (M, Oi, Os) \text{ sat prop NONE}\end{array}\]
Theorems for each transition are possible. For the most part, they are formed by copying and pasting the theorems above with a few keywords changed. They are repetitive and no new information is gained by showing them here. For completeness, however, they are included in the appendices and file folders as described in the introduction to this chapter.

8.2 ssmConductORP: Multiple Principals

ssmConductORP is an example of a SSM with more than one principal authorized to execute transitions among states. The diagram is shown in figure 8.2.

Other than the number of principals, ssmConductToORP follows the same structure as ssmPB described in the previous section.
8.2.1 Principals

The principals are defined in the datatype \textit{stateRole}. There are three principals including Omni.

\[
\text{stateRole} = \text{PlatoonLeader} | \text{PlatoonSergeant} | \text{Omni}
\]

8.2.2 States

States are defined as \textit{slState}. The names differ only slightly from the diagram but are straightforward.

\[
\text{slState} = \text{CONDUCT\_ORP} \\
| \text{SECURE} \\
| \text{ACTIONS\_IN} \\
| \text{WITHDRAW} \\
| \text{COMPLETE}
\]
8.2.3 Outputs

Outputs are named similarly to ssmPB outputs.

\[ slOutput = \text{ConductORP} \]
\[ \quad | \text{Secure} \]
\[ \quad | \text{ActionsIn} \]
\[ \quad | \text{Withdraw} \]
\[ \quad | \text{Complete} \]
\[ \quad | \text{unAuthenticated} \]
\[ \quad | \text{unAuthorized} \]

8.2.4 Commands

The \textit{slCommand} datatype defines three datatype variables, one for each principal. Each are further defined.

\[ slCommand = \text{PL plCommand} \]
\[ \quad | \text{PSG psgCommand} \]
\[ \quad | \text{OMNI omniCommand} \]

\textit{plCommand} defines the PlatoonLeader commands.

\[ plCommand = \text{secure} \]
\[ \quad | \text{withdraw} \]
\[ \quad | \text{complete} \]
\[ \quad | \text{plIncomplete} \]

\textit{psgCommand} defines the PlatoonSergeant commands.
\textit{psgCommand} = \textit{actionsIn} | \textit{psgIncomplete}

\textit{omniCommand} defines the Omni commands.

\textit{omniCommand} = \textit{ssmSecureComplete} \\
| \textit{ssmActionsInComplete} \\
| \textit{ssmWithdrawComplete} \\
| \textit{invalidOmniCommand}

\subsection*{8.2.5 Next-State Function}

The next-state function follows the same pattern as for \textit{ssmPB}. The only difference is that one of the transitions requires a \textit{psgCommand} instead of a \textit{plCommand}. This is the transition from \textit{SECURE} to \textit{ACTIONS-IN}.

\begin{verbatim}
[conductORPNS_def]

⊤ (conductORPNS CONDUCT_ORP (exec x) = \n  if getPlCom x = secure then SECURE else CONDUCT_ORP) \land 
  (conductORPNS SECURE (exec x) = \n  if getPsgCom x = actionsIn then ACTIONS_IN else SECURE) \land 
  (conductORPNS ACTIONS_IN (exec x) = \n  if getPlCom x = withdraw then WITHDRAW else ACTIONS_IN) \land 
  (conductORPNS WITHDRAW (exec x) = \n  if getPlCom x = complete then COMPLETE else WITHDRAW) \land 
  (conductORPNS s (trap x) = s) \land 
  (conductORPNS s (discard x) = s)

\end{verbatim}

The next-state function requires two helper functions just as the next-state function for \textit{ssmPB}. These are \textit{getPlCom} and \textit{getPsgCom}. The former extracts PlatoonLeader
commands and the later extracts PlatoonSergeant commands. They both use the "_" as a wild card. The HOL definition for both functions is shown below.

```haskell
val getPlCom_def = Define
  (getPlCom ([]:((slCommand command)option) list)
   = plIncomplete:plCommand) /
  (getPlCom (SOME (SLc (PL cmd)):(slCommand command)option::xs)
   = cmd:plCommand) /
  (getPlCom (_::(xs :(slCommand command)option list))
   = (getPlCom xs))'

val getPsgCom_def = Define
  (getPsgCom ([]:((slCommand command)option) list)
   = psgIncomplete:psgCommand) /
  (getPsgCom (SOME (SLc (PSG cmd)):(slCommand command)option::xs)
   = cmd:psgCommand) /
  (getPsgCom (_::(xs :(slCommand command)option list))
   = (getPsgCom xs))'
```

8.2.6 Next-Output Function

The next-output function follows the same pattern as the next-state function. It returns outputs instead of states.

```haskell
[conductORPOut_def]

| (conductORPOut CONDUCT_ORP (exec x)) =
|   if getPlCom x = secure then Secure else ConductORP) ∧
| (conductORPOut SECURE (exec x)) =
|   if getPsgCom x = actionsIn then ActionsIn else Secure) ∧
| (conductORPOut ACTIONS_IN (exec x)) =
|   if getPlCom x = withdraw then Withdraw else ActionsIn) ∧
| (conductORPOut WITHDRAW (exec x)) =
|   if getPlCom x = complete then Complete else Withdraw) ∧
```

125
\[
\begin{align*}
\text{(conductORPOut s (trap x) = unAuthorized)} & \land \\
\text{(conductORPOut s (discard x) = unAuthenticated)}
\end{align*}
\]

### 8.2.7 Authentication

Authentication uses the *inputOK* function. It is the same function as the ssmPB *inputOK* except that it adds an additional pattern matching to authenticate the Platoon Sergeant.

\[
\text{val inputOK_def =}
\]

Define

\[
\text{`(inputOK}
\]

\[
\text{((Name PlatoonLeader) says (prop (cmd:(slCommand command)option}))}
\]

\[
\text{(:,:,\text{(slCommand command)option, stateRole, }'d','e\text{)}Form) = T) } \land \\
\text{(inputOK}
\]

\[
\text{((Name PlatoonSergeant) says (prop (cmd:(slCommand command)option)))}
\]

\[
\text{(:,:,\text{(slCommand command)option, stateRole, }'d','e\text{)}Form) = T) } \land \\
\text{(inputOK}
\]

\[
\text{((Name Omni) says (prop (cmd:(slCommand command)option)))}
\]

\[
\text{(:,:,\text{(slCommand command)option, stateRole, }'d','e\text{)}Form) = T) } \land \\
\text{(inputOK _ = F)}`
\]

This function only allows the Platoon Leader, Platoon Sergeant, and Omni as authenticated entities for state-level commands. Furthermore, it restricts them from being authenticated on any other commands. It also denies authentication to any other principal on any other command.

It is straightforward to prove that any command that is not issued by a principal is rejected. This follows directly from the definition of *inputOK*.

\[\text{[inputOK\_cmd\_reject\_lemma]}\]

\[\vdash \forall \text{cmd}. \neg \text{inputOK (prop (SOME cmd))}\]
8.2.8 Authorization

As in ssmPB and all SSMs, there is a state-dependent and state-independent authorization function.

**State-dependent Authorization** Like ssmPB, the state-dependent authorization function is named `secContext`. It takes a state and an input list as parameters. It returns an authorization statement. It follows the same logic as `secContext` in ssmPB.

```plaintext
[secContext_def]

\[ \exists (\forall xs. \\
\quad \text{secContext CONDUCT_ORP } xs = \\
\quad \quad \text{[Name PlatoonLeader controls} \\
\quad \quad \quad \text{prop (SOME (SLc (PL secure)))]} \land \\
\quad \quad (\forall xs. \\
\quad \quad \text{secContext SECURE } xs = \\
\quad \quad \quad \text{if getOmniCommand } xs = \text{ssmSecureComplete then} \\
\quad \quad \quad \quad \quad \text{[prop (SOME (SLc (OMNI ssmSecureComplete))} \text{ impf} \\
\quad \quad \quad \quad \quad \text{Name PlatoonSergeant controls} \\
\quad \quad \quad \quad \quad \text{prop (SOME (SLc (PSG actionsIn))))]} \\
\quad \quad \quad \text{else [prop NONE}) \land \\
\quad \quad (\forall xs. \\
\quad \quad \text{secContext ACTIONS_IN } xs = \\
\quad \quad \quad \text{if getOmniCommand } xs = \text{ssmActionsInComplete then} \\
\quad \quad \quad \quad \quad \text{[prop (SOME (SLc (OMNI ssmActionsInComplete))} \text{ impf} \\
\quad \quad \quad \quad \quad \text{Name PlatoonLeader controls} \\
\quad \quad \quad \quad \quad \text{prop (SOME (SLc (PL withdraw))))]} \\
\quad \quad \quad \text{else [prop NONE}) \land \\
\forall xs. \]
```
secContext WITHDRAW xs =
  if getOmniCommand xs = ssmWithdrawComplete then
    [prop (SOME (SLc (OMNI ssmWithdrawComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL complete)))]
  else [prop NONE]

secContext in ssmConductORP uses the exact same getOmniCommand function. It is repeated here as a reference.

val getOmniCommand_def =
Define
  (getOmniCommand ([]:((slCommand command)option , stateRole , 'd','e)Form list)
    = invalidOmniCommand:omniCommand) \ /
  (getOmniCommand (((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs)
    = (cmd:omniCommand)) \ /
  (getOmniCommand ((x:((slCommand command)option , stateRole , 'd','e)Form)::xs)
    = (getOmniCommand xs))'

State-independent Authorization  Like ssmPB, the state-independent authorization function is named secAuthorization. It takes only an input list as a parameter. This is the exact same secAuthorization as in ssmPB. It also uses the exact same secHelper function as ssmPB. It is repeated here as a reference.

[secAuthorization_def]

⊢ ∀xs. secAuthorization xs = secHelper (getOmniCommand xs)

[secHelper_def]

⊢ ∀cmd.

  secHelper cmd =
    [Name Omni controls prop (SOME (SLc (OMNI cmd)))]
8.2.9 Proved Theorems

These theorems follow those in section 8.1.9 for ssmPB. There are few changes and thus detailed explanation of the proof is omitted. The pretty-printed HOL-generated output is included here.

**PlatoonSergeant_SECURE_exec_justified_thm**  This theorem justifies transition from the SECURE state to the ACTIONS_IN state. The authenticated principal is the PlatoonSergeant. This theorem specializes the \( TR_{exec\_cmd\_rule} \). The two lemmas and theorem are shown below.

\[
\text{PlatoonSergeant_SECURE_exec_lemma}
\]

\[
\vdash \forall M \ Oi \ Os. \\
\text{CFGInterpret} \ (M, Oi, Os) \\
\text{propCommandList}
\]

\[
\text{PlatoonSergeant_SECURE_exec_justified_lemma}
\]
∀ NS Out M Oi Os.

TR (M, Oi, Os)

(exec

(inputList

[Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
  prop (SOME (SLc (PSG actionsIn))))]

(CFG inputOK secContext secAuthorization

([Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
  prop (SOME (SLc (PSG actionsIn))))::ins) SECURE outs)

(CFG inputOK secContext secAuthorization ins

(NS SECURE

(exec

(inputList

[Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
  prop (SOME (SLc (PSG actionsIn))))]

(Out SECURE

(exec

(inputList

[Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
  prop (SOME (SLc (PSG actionsIn))))::

outs)) ↔

authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete))];
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))] ^

CFGInterpret \( M, O_i, O_s \)

\( \text{CFG inputOK secContext secAuthorization} \)

\( ([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete))]);
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))] :: ins) \text{ SECURE outs} \) ^

\( (M, O_i, O_s) \text{ satList} \)
propCommandList

[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]

\[\text{PlatoonSergeant\_SECURE\_exec\_justified\_thm}\]

\( \vdash \forall NS \text{ Out } M O_i O_s . \)

\( \text{TR} \ (M, O_i, O_s) \)

\( \text{exec} \)

\( \text{[SOME (SLc (OMNI ssmSecureComplete))]; SOME (SLc (PSG actionsIn))]} \)

\( \text{CFG inputOK secContext secAuthorization} \)

\( ([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete))]);
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))] :: ins) \text{ SECURE outs} \)

131
PlatoonLeader_ACTIONS_IN_exec_justified_thm  This theorem justifies transition from the ACTIONS_IN state to the WITHDRAW state. The authenticated principal is the PlatoonLeader. This theorem specializes the TR_exec_cmd_rule. The two lemmas and theorem are shown below.
∀ M Oi Os.

CFGInterpret (M, Oi, Os)

(CFG inputOK secContext secAuthorization

([Name Omni says
    prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
    prop (SOME (SLc (PL withdraw)))]::ins) ACTIONS_IN

outs) ⇒

(M, Oi, Os) satList

propCommandList

[Name Omni says
    prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]

∀ NS Out M Oi Os.

TR (M, Oi, Os)

(exec

(inputList

[Name Omni says
    prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
    prop (SOME (SLc (PL withdraw)))]))

(CFG inputOK secContext secAuthorization

([Name Omni says
    prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
    prop (SOME (SLc (PL withdraw)))]::ins) ACTIONS_IN
(CFG inputOK secContext secAuthorization ins

(NS ACTIONS_IN
(exec
(inputList
[Name Omni says
  prop
    (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
  prop (SOME (SLc (PL withdraw)))]))

(Out ACTIONS_IN
(exec
(inputList
[Name Omni says
  prop
    (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
  prop (SOME (SLc (PL withdraw)))]):

outs)) \iff

authenticationTest inputOK
[Name Omni says
  prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
  prop (SOME (SLc (PL withdraw)))] \land

CFGInterpret \((M, O_i, O_s)\)
(CFG inputOK secContext secAuthorization
  (\[[Name Omni says
    prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
    prop (SOME (SLc (PL withdraw)))] : ins) ACTIONS_IN
outs) \land
(M, Oi, Os) satList
propCommandList

[Name Omni says
prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]

[PlatoonLeader_ACTIONS_IN_exec_justified_thm]

\[\forall NS \ Out \ M \ Oi \ Os.\]

TR (M, Oi, Os)

(exec
[SOME (SLc (OMNI ssmActionsInComplete))];
SOME (SLc (PL withdraw))])

(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))::ins) ACTIONS_IN outs)

(CFG inputOK secContext secAuthorization ins

(\NS \ ACTIONS_IN
(exec
[SOME (SLc (OMNI ssmActionsInComplete))];
SOME (SLc (PL withdraw))]))

(Out \ ACTIONS_IN
(exec
[SOME (SLc (OMNI ssmActionsInComplete))];
SOME (SLc (PL withdraw)))::outs)) \iff
authenticationTest inputOK

[Name Omni says
prop (SOME (SLc (OMNI ssmActionsInComplete)))]
The above theorem formally proves the soundness of the CONOPs with respect to the Platoon Leader’s command to withdraw from the ACTIONS_IN state. This theorem proves that execution of the transition from ACTIONS_IN to the next state happens if and only if the Platoon Leader issues the command ssmActionsInComplete.

PlatoonLeader_ACTIONS_IN_trap_justified_thm This theorem justifies trapping the transition from the ACTIONS_IN state to the WITHDRAW state. The authenticated principal is the PlatoonLeader. But, in this case, Omni does not issue the command ssmActionsInComplete. Therefore, the transition is not authorized. This theorem specializes the TR_trap_cmd_rule. The two lemmas and theorem are shown below.

[PlatoonLeader_ACTIONS_IN_trap_lemma]

\[ \Downarrow \text{omniCommand} \neq \text{ssmActionsInComplete} \Rightarrow \]

\[ (s = \text{ACTIONS\_IN}) \Rightarrow \]

\[ \forall M \ Oi \ Os . \]
CFGInterpret \((M, Oi, Os)\)

\[
\begin{align*}
\text{CFG inputOK secContext secAuthorization} \\
([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \\
\text{Name PlatoonLeader says} \\
\text{prop (SOME (SLc (PL withdraw))))]}::ins) \text{ ACTIONS_IN} \\
\text{outs} \Rightarrow \\
(M, Oi, Os) \text{ sat prop NONE}
\end{align*}
\]

\[\text{PlatoonLeader\_ACTIONS\_IN\_trap\_justified\_lemma}\]

\[\vdash \text{omniCommand} \neq \text{ssmActionsInComplete} \Rightarrow \]
\[\quad (s = \text{ACTIONS\_IN}) \Rightarrow \]
\[\forall NS \text{ Out M Oi Os.} \]
\[\text{TR} \ (M, Oi, Os) \]
\[\quad (\text{trap} \]
\[\quad \quad (\text{inputList} \]
\[\quad \quad \quad [\text{Name Omni says} \]
\[\quad \quad \quad \quad \text{prop (SOME (SLc (OMNI omniCommand)))}; \\
\quad \quad \quad \text{Name PlatoonLeader says} \\
\quad \quad \quad \text{prop (SOME (SLc (PL withdraw)))))})
\]
\[\text{CFG inputOK secContext secAuthorization} \]
\[\quad ([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \\
\quad \text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL withdraw))))]}::ins) \text{ ACTIONS_IN} \\
\quad \text{outs}) \]
\[\text{CFG inputOK secContext secAuthorization ins} \]
\[\quad (NS \text{ ACTIONS\_IN} \]
\[\quad \quad (\text{trap} \]
\[\quad \quad \quad (\text{inputList} \]
\[\quad \quad \quad \quad [\text{Name Omni says} \\
\quad \quad \quad \quad \text{prop (SOME (SLc (OMNI omniCommand)))}; \\
\]
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))

(Out ACTIONS_IN
(trap
(inputList
[Name Omni says
prop (SOME (SLc (OMNI omniCommand))));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))::
out)
authenticationTest inputOK
[Name Omni says prop (SOME (SLc (OMNI omniCommand))));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw)))) ∨

CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand))));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))::ins) ACTIONS_IN
outs) ∨ (M, Oi, Os) sat prop NONE

[PlatoonLeader_ACTIONS_IN_trap_justified_thm]

\[ omniCommand \neq ssmActionsIncomplete \Rightarrow \\
\begin{equation}
(s = ACTIONS_IN) \Rightarrow \\
\forall NS \ Out M Oi Os.
\end{equation}

TR (M, Oi, Os)
(trap
[SOME (SLc (OMNI omniCommand)));
SOME (SLc (PL withdraw))])
(CFG inputOK secContext secAuthorization

138
As with ssmPlanPB, the other theorems are included in the appendices and files.
8.3 ssmPlanPB: Non-sequential Transitions

The ssmPlanPB SSM does not use the Omni principal but does use a "sort-of" non-sequential progression through states. To minimize the complexity of the model, the three states that are non-sequential are represented as "virtual states" whose completion is noted as input to the command to transition from the state before them to the state after them. Figure 8.3 is a diagram of this SSM.

The three non-sequential states are hidden in the white circle in the diagram. They are TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON. These three states may be completed in any order, but all three must be completed before progressing to the next state REPORT1.

To handle this, the three non-sequential states are combined into one "virtual state." Completion of these states must preceded transition from the WARNNO state to the REPORT1 state. Completion is indicated by the following three ACL statements

PlatoonLeader says tentativePlan

PlatoonSergeant says initiateMovement

PlatoonLeader says recon

Figure 8.3: Horizontal slice: PlanPB diagram.
when combined with a request from the PlatoonLeader to transition to the REPORT1 state, the input (a list of inputs) for this transition is

\[
\text{PlatoonLeader says tentativePlan}
\]

\[
\text{PlatoonSergeant says initiateMovement}
\]

\[
\text{PlatoonLeader says recon}
\]

\[
\text{PlatoonLeader says report1}
\]

The security policy handles this with the following implication

\[
tentativePlan \text{ andf}
\]

\[
initiateMovement \text{ andf}
\]

\[
\text{recon impf}
\]

\[
\text{PlatoonLeader controls recon}
\]

The remaining details of this implementation follow.

### 8.3.1 Principals

The planning phase SSM has two principals: PlatoonLeader and PlatoonSergeant. These are defined in the stateRole datatype.

\[
\text{stateRole} = \text{PlatoonLeader} \mid \text{PlatoonSergeant}
\]
8.3.2 States

There are 12 states. But, TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON are virtual states and not used in the next-state and next-output functions.

\[ slState = PLAN_PB \]
\[ \quad | \text{RECEIVE MISSION} \]
\[ \quad | \text{WARNO} \]
\[ \quad | \text{TENTATIVE_PLAN} \]
\[ \quad | \text{INITIATE_MOVEMENT} \]
\[ \quad | \text{RECON} \]
\[ \quad | \text{REPORT1} \]
\[ \quad | \text{COMPLETE_PLAN} \]
\[ \quad | \text{OPOID} \]
\[ \quad | \text{SUPERVISE} \]
\[ \quad | \text{REPORT2} \]
\[ \quad | \text{COMPLETE} \]

8.3.3 Output

There are 14 outputs. But, PlanPB, TentativePlan, InitiateMovement, and Recon are not used in the next-output function. The unAuthorized output is returned for trapped commands. The unAuthenticated output is returned for discarded commands.

\[ slOutput = PlanPB \]
\[ \quad | \text{ReceiveMission} \]
\[ \quad | \text{Warno} \]
\[ \quad | \text{TentativePlan} \]
8.3.4 Commands

The \textit{slCommand} datatype for this SSM is defined below.

\[
\text{slCommand} = \text{PL plCommand} \mid \text{PSG psgCommand}
\]

The two datatypes for \textit{plCommand} and \textit{psgCommand} represent the PlatoonLeader and PlatoonSergeant commands, respectively. These are defined below.

\[
\text{plCommand} = \text{receiveMission} \\
\quad \mid \text{warno} \\
\quad \mid \text{tentativePlan} \\
\quad \mid \text{recon} \\
\quad \mid \text{report1} \\
\quad \mid \text{completePlan} \\
\quad \mid \text{opoid} \\
\quad \mid \text{supervise} \\
\quad \mid \text{report2}
\]

\[
\text{psgCommand} = \text{warno} \\
\quad \mid \text{tentativePlan} \\
\quad \mid \text{recon} \\
\quad \mid \text{report2} \\
\quad \mid \text{completePlan} \\
\quad \mid \text{opoid} \\
\quad \mid \text{supervise} \\
\quad \mid \text{report1}
\]

\[
\text{unAuthenticated} \mid \text{unAuthorized}
\]
Providing each principal with her own set of commands simplifies the authentication and authorization functions.

### 8.3.5 Next-State Function

The next-state function is defined similarly to those defined for previously described SSMs.

\[\text{planPBNS}_{\text{def}}\]

\[
\vdash \text{planPBNS \ WARNO (exec \( x \)) =}
\]

\[
\text{if } \left( \begin{array}{l}
\text{getRecon } x = \text{[SOME (SLc (PL recon))] } \land \\
\text{getTenativePlan } x = \text{[SOME (SLc (PL tentativePlan))] } \land \\
\text{getReport } x = \text{[SOME (SLc (PL report1))] } \land \\
\text{getInitMove } x = \text{[SOME (SLc (PSG initiateMovement))]} \\
\end{array} \right) \\
\text{then}
\]

\[
\text{REPORT1}
\]

\[
\text{else \ WARNO} \land \\
\text{(planPBNS \ PLAN\_PB (exec \( x \)) =}
\]

\[
\text{if getPlCom } x = \text{receiveMission then \ RECEIVE\_MISSION}
\]
else PLAN_PB) ∧
(planPBNS RECEIVE_MISSION (exec x) =
    if getPlCom x = warno then WARNO else RECEIVE_MISSION) ∧
(planPBNS REPORT1 (exec x) =
    if getPlCom x = completePlan then COMPLETE_PLAN
else REPORT1) ∧
(planPBNS COMPLETE_PLAN (exec x) =
    if getPlCom x = opoid then OPOID else COMPLETE_PLAN) ∧
(planPBNS OPOID (exec x) =
    if getPlCom x = supervise then SUPERVISE else OPOID) ∧
(planPBNS SUPERVISE (exec x) =
    if getPlCom x = report2 then REPORT2 else SUPERVISE) ∧
(planPBNS REPORT2 (exec x) =
    if getPlCom x = complete then COMPLETE else REPORT2) ∧
(planPBNS s (trap v₀) = s) ∧ (planPBNS s (discard v₁) = s)

Five functions are defined (see below) to extract the command from the input list.
These are getRecon, getTentativePlan, getReport, getInitMove, and getPlCom. Each of
these functions takes an input list as a parameter and returns a command with its
option type. For the sake of brevity, only getRecon is shown below. The other functions
are similar.

HOL Definition for getRecon ...

val getRecon_def = Define ' 

    (getRecon []:((s l Command command)option , stateRole , 'd','e)Form list) = 
       [NONE]) /

    (getRecon ((Name PlatoonLeader) says (prop (SOME (SLc (PL recon))))
       :((s l Command command)option , stateRole , 'd','e)Form::xs)
       = [SOME (SLc (PL recon)):(s l Command command)option]) /

    (getRecon (_::xs) = getRecon xs)"
8.3.6 Next-Output Function

The next-output function is defined similarly to those defined for previously described SSMs.

\[
\text{[planPBOOut\_def]}
\]

\[\vdash (\text{planPBOOut \text{\textsc{warno}} (exec } x) =
\]

\[\text{if (getRecon } x = [\text{SOME (SLc (PL recon))}] \land
\text{getTenativePlan } x = [\text{SOME (SLc (PL tentativePlan))}] \land
\text{getReport } x = [\text{SOME (SLc (PL report1))}] \land
\text{getInitMove } x = [\text{SOME (SLc (PSG initiateMovement))}]\]

\[\text{then Report1}
\]

\[\text{else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{plan\_pb}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{receiveMission then ReceiveMission}
\]

\[\text{else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{receive\_mission}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{warno then Warno else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{report\_1}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{completePlan then CompletePlan}
\]

\[\text{else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{complete\_plan}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{opoid then Opid else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{supervise}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{supervise then Supervise}
\]

\[\text{else } \text{unAuthorized}) \land
\]

\[\text{(planPBOOut \text{\textsc{supervise}} (exec } x) =
\]

\[\text{if getPlCom } x = \text{report2 then Report2 else } \text{unAuthorized}) \land
\]
\begin{verbatim}(planPBOut REPORT2 (exec x) =
   if getPlCom x = complete then Complete else unAuthorized) \land
   (planPBOut s (trap v_0) = unAuthorized) \land
   (planPBOut s (discard v_1) = unAuthenticated)
\end{verbatim}

8.3.7 Authentication

Authentication for this SSM is the same as for the previously described SSMs.

\begin{verbatim}val inputOK_def = Define
  `(inputOK
    (((Name PlatoonLeader) says (prop (cmd:((slCommand command)option))))
     :((slCommand command)option, stateRole, ’d,’e)Form) = T) \land
    (inputOK
     (((Name PlatoonSergeant) says (prop (cmd:((slCommand command)option))))
      :((slCommand command)option, stateRole, ’d,’e)Form) = T) \land
    (inputOK _ = F)`
\end{verbatim}

It is straightforward to prove that any command that is not requested by a principal is false.

\begin{verbatim}val inputOK_cmd_reject_lemma =
TAC_PROOF(
  [],
  `!cmd. ~(inputOK
    ((prop (SOME cmd)):((slCommand command)option, stateRole, ’d,’e)Form))`,
  PROVE_TAC[ inputOK_def ])
\end{verbatim}

8.3.8 Authorization

The two authorization functions are defined below.

\begin{verbatim}val secContext_def = Define `
secContext (s:slState) (x:((slCommand command)option, stateRole, 'd','e)Form list) =
    if (s = WARNO) then
        (if (getRecon x = SOME (SLc PL recon))
            (if (getTenativePlan x = SOME (SLc PL tentativePlan))
                (if (getReport x = SOME (SLc PL report1))
                    (if (getInitMove x = SOME (PSG initiateMovement))
                        then PL_WARNO_Auth
                            (Name PlatoonLeader) controls prop (SOME (SLc PL recon));
                            (Name PlatoonLeader) controls prop (SOME (SLc PL tentativePlan));
                            (Name PlatoonSergeant) controls prop (SOME (PSG initiateMovement))
                    else ((prop NONE):((slCommand command)option, stateRole, 'd','e)Form))
                else ((prop NONE):((slCommand command)option, stateRole, 'd','e)Form))
            else if ((getPlCom x) = invalidPlCommand)
                then ((prop NONE):((slCommand command)option, stateRole, 'd','e)Form)
            else [PL_notWARNO_Auth (getPlCom x)])
    else [PL_notWARNO_Auth (cmd)]

secContext uses PL_WARNO_Auth and PL_notWARNO_Auth. These are defined below.

val PL_WARNO_Auth_def = Define `  
  PL_WARNO_Auth =
      [\((impTermList
          [(prop (SOME (SLc PL recon)))]
          (Name PlatoonLeader) controls prop (SOME (SLc (PL tentativePlan))))
          (Name PlatoonSergeant) controls prop (SOME (PSG initiateMovement))))
  `]

val PL_notWARNO_Auth_def = Define `  
  PL_notWARNO_Auth (cmd:plCommand) =
      if (cmd = report1) (* report1 exits WARNO state *)
      then (prop NONE):((slCommand command)option, stateRole, 'd','e)Form
      else (Name PlatoonLeader) says (prop (SOME (SLc (PL cmd)))]
8.3.9 Proved Theorems

Theorems proving complete mediation follow the same format as those for previously described SSMs.

PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm This theorem proves that if the state is not WARNO and the plCommand is not report1 then execution of any plCommand is justified. It uses two lemmas and a theorem and follows the same pattern as the the other three-part proofs. The pretty-printed HOL-generated output is shown below.

PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma

\[ \vdash s \neq \text{WARNO} \Rightarrow \\
\quad \text{plCommand} \neq \text{invalidPlCommand} \Rightarrow \\
\quad \text{plCommand} \neq \text{report1} \Rightarrow \\
\quad \forall M\ Oi\ Os. \\
\quad \text{CFGInterpret} (M, Oi, Os) \
\quad \text{(CFG inputOK secContext secContextNull} \\
\quad \quad \text{([Name PlatoonLeader says} \\
\quad \quad \quad \text{prop (SOME (SLc (PL plCommand)))]}::\text{ins}) \ s\ outs) \Rightarrow \\
\quad \ (M, Oi, Os)\ \text{satList} \]
propCommandList

[Name PlatoonLeader says
 prop (SOME (SLc (PL plCommand)))]

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]

\[ s \neq \text{WARNO} \Rightarrow \]

\[ p\text{Command} \neq \text{invalidPlCommand} \Rightarrow \]

\[ p\text{Command} \neq \text{report1} \Rightarrow \]

\[ \forall NS \ Out M Oi Os. \]

TR (M, Oi, Os)

(exec

(inputList

[Name PlatoonLeader says
 prop (SOME (SLc (PL plCommand)))]

(CFG inputOK secContext secContextNull

([Name PlatoonLeader says
 prop (SOME (SLc (PL plCommand))))::ins) s outs)

(CFG inputOK secContext secContextNull ins

(NS s

(exec

(inputList

[Name PlatoonLeader says
 prop (SOME (SLc (PL plCommand)))]

(Out s

(exec

(inputList

[Name PlatoonLeader says
 prop (SOME (SLc (PL plCommand)))]::

outs)) \iff 

150
authenticationTest inputOK

[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand))) ] ∧

CFGInterpret (M, Oi, Os)

(CFG inputOK secContext secContextNull
  ( [Name PlatoonLeader says
     prop (SOME (SLc (PL plCommand)))) :: ins ] s outs ) ∧
  (M, Oi, Os) satList
  propCommandList
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand))) ]

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

\[ s \neq \text{WARNO} \Rightarrow \]

\[ pl\text{Command} \neq \text{invalidPlCommand} \Rightarrow \]

\[ pl\text{Command} \neq \text{report1} \Rightarrow \]

\[ \forall NS \text{ Out } M Oi Os . \]

TR (M, Oi, Os) (exec [SOME (SLc (PL plCommand))])

(CFG inputOK secContext secContextNull
  ( [Name PlatoonLeader says
     prop (SOME (SLc (PL plCommand)))) :: ins ] s outs )

(CFG inputOK secContext secContextNull ins
  (NS s (exec [SOME (SLc (PL plCommand))])))

(Out s (exec [SOME (SLc (PL plCommand))]) :: outs) \iff

authenticationTest inputOK

[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand))) ] ∧

CFGInterpret (M, Oi, Os)

(CFG inputOK secContext secContextNull
  ( [Name PlatoonLeader says
     prop (SOME (SLc (PL plCommand))) ]
prop (SOME (SLc (PL plCommand))) :: ins) s outs) ∧
(M, Oi, Os) satList [prop (SOME (SLc (PL plCommand)))]

**PlatoonLeader_psgCommand_notDiscard_thm**  This next theorem proves that if the PlatoonLeader issues a *psgCommand* then that command is not discarded. The reason for this is that the *psgCommand* is an *slCommand*. In *inputOK*, the PlatoonLeader is authenticated on all *slCommands*. (Only unauthenticated requests are discarded.) This theorem has no lemmas.

[PlatoonLeader_psgCommand_notDiscard_thm]

\[ ∀ NS \ Out \ M \ Oi \ Os .
\neg \text{TR} (M, Oi, Os) (\text{discard} [\text{SOME} (\text{SLc} (\text{PSG psgCommand})))]) \]
(CFG inputOK secContext secContextNull
([Name PlatoonLeader says
prop (SOME (SLc (PSG psgCommand))) :: ins) s outs)

(CFG inputOK secContext secContextNull ins
(NS s (discard [SOME (SLc (PSG psgCommand))])))
(Out s (discard [SOME (SLc (PSG psgCommand))]) :: outs))

**PlatoonSergeant_trap_plCommand_justified_thm**  This next theorem proves that if the PlatoonLeader issues a *psgCommand* then that command is trapped. It specializes the *TR_trap_cmd_rule* with two lemmas and a theorem.

[PlatoonLeader_trap_psgCommand_lemma]

\[ ∀ M \ Oi \ Os .
\text{CFGInterpret} (M, Oi, Os)

(CFG inputOK secContext secContextNull

152
([Name PlatoonLeader says
   prop (SOME (SLc (PSG psgCommand)))]::ins) s outs) ⇒
(M, Oi, Os) sat prop NONE

[PlatoonLeader_trap_psgCommand_justified_lemma]

\[ ∀ NS \ Out \ M \ Oi \ Os . \]
\[ TR (M, Oi, Os) \]
\[ (trap \]
\[ (inputList
   [Name PlatoonLeader says
      prop (SOME (SLc (PSG psgCommand))))]
)
(CFG inputOK secContext secContextNull
([Name PlatoonLeader says
   prop (SOME (SLc (PSG psgCommand)))]::ins) s outs)
(CFG inputOK secContext secContextNull ins
(NS s
(trap
(trap
(inputList
[Name PlatoonLeader says
   prop (SOME (SLc (PSG psgCommand))))]))

(Out s
(trap
(trap
(inputList
[Name PlatoonLeader says
   prop (SOME (SLc (PSG psgCommand))))]
outs)) \iff\]
authenticationTest inputOK
[Name PlatoonLeader says
   prop (SOME (SLc (PSG psgCommand)))] \land
CFGInterpret (M, Oi, Os)

153
\[
\vdash \forall NS\ Out\ M\ Oi\ Os.
\]

\[
\text{TR}\ (M, Oi, Os)\ (\text{trap}\ [\text{SOME}\ (\text{SLc}\ (\text{PL} \ plCommand))])
\]

\[
\text{(CFG inputOK secContext secContextNull}
\]

\[
\text{([Name PlatoonSergeant says}
\]

\[
\text{prop\ (SOME\ (SLc\ (PL \ plCommand)))
}\]

\[
\text{::ins}) s\ outs})\ ^
\]

\[
\text{(M, Oi, Os) sat prop NONE}
\]

\[
\text{PlatoonSergeant_trap_plCommand_justified_thm}
\]

\[
\text{PlatoonLeader_WARNO_exec_report1_justified_thm}\quad \text{This theorem proves that transition from the WARNO to the REPORT1 is justified if the following four statements are supplied.}
\]

\begin{itemize}
  \item PlatoonLeader says tentativePlan
  \item PlatoonSergeant says initiateMovement
\end{itemize}
• PlatoonLeader says recon
• PlatoonLeader says report1

The above theorem formally proves the soundness of the CONOPs with respect to the Platoon Leader’s command to transition from the WARNO state to the REPORT1 state. This theorem proves that execution of this transition happens if and only if the following four commands are issued: PlatoonLeader says tentativePlan, PlatoonSergeant says initiateMovement, PlatoonLeader says recon, and PlatoonLeader says report1.

It specializes the \( TR_{\text{exex\_cmd\_rule}} \) and requires the standard two lemmas and a theorem.

\[\text{PlatoonLeader\_WARNO\_exec\_report1\_lemma}\]

\[\vdash \forall M\ Oi\ Os. \]
\[\text{CFGInterpret } (M,Oi,Os) \]
\[\text{(CFG inputOK secContext secContextNull } \]
\[\text{([Name PlatoonLeader says} \]
\[\text{prop (SOME (SLc (PL recon))));} \]
\[\text{Name PlatoonLeader says} \]
\[\text{prop (SOME (SLc (PL tentativePlan))));} \]
\[\text{Name PlatoonSergeant says} \]
\[\text{prop (SOME (SLc (PSG initiateMovement))));} \]
\[\text{Name PlatoonLeader says} \]
\[\text{prop (SOME (SLc (PL report1))))])::ins) \text{ WARNO outs } \Rightarrow \]
\[\text{(}M,Oi,Os\) satList} \]
\[\text{propCommandList} \]
\[\text{[Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[\text{Name PlatoonLeader says} \]
\[\text{prop (SOME (SLc (PL tentativePlan))));} \]
\[\text{Name PlatoonSergeant says} \]

155
prop (SOME (SLc (PSG initiateMovement)))
Name PlatoonLeader says prop (SOME (SLc (PL report1)))

PlatoonLeader_WARNO_exec_report1_justified_lemma

\[ \forall NS \ Out M \ Oi \ Os. \]
\[ \text{TR} (M, Oi, Os) \]
\[ (\text{exec} \]
\[ (\text{inputList} \]
\[ [\text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL recon)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL tentativePlan)))}; \]
\[ \text{Name PlatoonSergeant says} \]
\[ \text{prop (SOME (SLc (PSG initiateMovement)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL report1))))]) \]
\[ (\text{CFG inputOK secContext secContextNull} \]
\[ ([\text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL recon)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL tentativePlan)))}; \]
\[ \text{Name PlatoonSergeant says} \]
\[ \text{prop (SOME (SLc (PSG initiateMovement)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL report1))))])::ins) \text{ WARNO outs} \]
\[ (\text{CFG inputOK secContext secContextNull} ins \]
\[ (NS \text{ WARNO} \]
\[ (\text{exec} \]
\[ (\text{inputList} \]
\[ [\text{Name PlatoonLeader says} \]

156
prop (SOME (SLc (PL recon)))
Name PlatoonLeader says
prop (SOME (SLc (PL tentativePlan)))
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)))
Name PlatoonLeader says
prop (SOME (SLc (PL report1))))))}

(Out WARNO
(exec
(inputList
[Name PlatoonLeader says
prop (SOME (SLc (PL recon)))
Name PlatoonLeader says
prop (SOME (SLc (PL tentativePlan)))
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)))
Name PlatoonLeader says
prop (SOME (SLc (PL report1))))::outs)) \leftrightarrow
authenticationTest inputOK
[Name PlatoonLeader says prop (SOME (SLc (PL recon)))];
Name PlatoonLeader says
prop (SOME (SLc (PL tentativePlan)))
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)))
Name PlatoonLeader says
prop (SOME (SLc (PL report1))))] \land

CFGInterpret (M, O_i, O_s)
(CFG inputOK secContext secContextNull
)([Name PlatoonLeader says
prop (SOME (SLc (PL recon)))];
Name PlatoonLeader says

157
prop (SOME (SLc (PL tentativePlan)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)));
Name PlatoonLeader says
prop (SOME (SLc (PL report1)))::ins) WARNO outs) \wedge
(M, Oi, Os) satList
propCommandList

[Name PlatoonLeader says prop (SOME (SLc (PL recon)));
Name PlatoonLeader says
prop (SOME (SLc (PL tentativePlan)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)));
Name PlatoonLeader says prop (SOME (SLc (PL report1))))]

[PlatoonLeader_WARNO_exec_report1_justified_thm]

\( \forall NS \ Out \ M \ Oi \ Os. \)

TR (M, Oi, Os)

(exec
[SOME (SLc (PL recon)); SOME (SLc (PL tentativePlan));
SOME (SLc (PSG initiateMovement));
SOME (SLc (PL report1))])

(CFG inputOK secContext secContextNull

{{Name PlatoonLeader says
prop (SOME (SLc (PL recon)));
Name PlatoonLeader says
prop (SOME (SLc (PL tentativePlan)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)));
Name PlatoonLeader says
prop (SOME (SLc (PL report1))))::ins) WARNO outs)

158
(CFG inputOK secContext secContextNull \( ins \)

\( \textbf{NS} \ \text{WARNO} \)

\{exec

[SOME (SLc (PL recon));
SOME (SLc (PL tentativePlan));
SOME (SLc (PSG initiateMovement));
SOME (SLc (PL report1))]);

\( \textbf{Out} \ \text{WARNO} \)

\{exec

[SOME (SLc (PL recon));
SOME (SLc (PL tentativePlan));
SOME (SLc (PSG initiateMovement));
SOME (SLc (PL report1))]); \( :: \text{outs} \)\)

\( \iff \)

authenticationTest inputOK

\[\text{Name PlatoonLeader says prop (SOME (SLc (PL recon))));\]
\text{Name PlatoonLeader says}
\text{prop (SOME (SLc (PL tentativePlan)));}
\text{Name PlatoonSergeant says}
\text{prop (SOME (SLc (PSG initiateMovement)));}
\text{Name PlatoonLeader says}
\text{prop (SOME (SLc (PL report1)))] \land}

\( \text{CFGInterpret} \ (M, Oi, Os) \)

\( \text{(CFG inputOK secContext secContextNull \( ::ins\) WARNO \( \text{outs} \) \land} \)
(\(M, Oi, Os\)) satList

\[
\begin{aligned}
&\text{prop (SOME (SLc (PL recon)))}; \\
&\text{prop (SOME (SLc (PL tentativePlan)))}; \\
&\text{prop (SOME (SLc (PSG initiateMovement)))}; \\
&\text{prop (SOME (SLc (PL report1)))}
\end{aligned}
\]

**PlatoonSergeant_trap_plCommand_justified_thm** This final theorem proves that the PlatoonSergeant is trapped on all \(pl\text{Commands} \). This is for the same reason that the PlatoonLeader is trapped on all \(psg\text{Commands} \).

**PlatoonSergeant_trap_plCommand_lemma**

\[\forall M \ Oi \ Os. \]

\[
\begin{aligned}
&\text{CFGInterpret \((M, Oi, Os)\)} \\
&(\text{CFG inputOK secContext secContextNull} \\
&\quad ([\text{Name PlatoonSergeant says} \\
&\quad \quad \text{prop (SOME (SLc \ (PL plCommand))}: ins} \ s \ outs) \Rightarrow \\
&\quad \quad (M, Oi, Os) \ \text{sat prop NONE})
\end{aligned}
\]

**PlatoonSergeant_trap_plCommand_justified_lemma**

\[\forall NS \ Out \ M \ Oi \ Os. \]

\[
\begin{aligned}
&\text{TR \((M, Oi, Os)\)} \\
&(\text{trap} \\
&\quad \text{(inputList} \\
&\quad \quad \text{[Name PlatoonSergeant says} \\
&\quad \quad \quad \text{prop (SOME (SLc \ (PL plCommand))}): ins} \ s \ outs}) \\
&\quad \text{(CFG inputOK secContext secContextNull} \\
&\quad \quad ([\text{Name PlatoonSergeant says} \\
&\quad \quad \quad \text{prop (SOME (SLc \ (PL plCommand))}): ins} \ s \ outs) \\
&\quad \quad \text{(CFG inputOK secContext secContextNull ins})
\end{aligned}
\]
\[(NS\ s\n\quad (\text{trap}\n\quad (\text{inputList}\n\quad [\text{Name PlatoonSergeant says prop (SOME (SLc (PL pl\text{Command}))))}))\n\quad (Out\ s\n\quad (\text{trap}\n\quad (\text{inputList}\n\quad [\text{Name PlatoonSergeant says prop (SOME (SLc (PL pl\text{Command}))))])))\n\quad outs)) \iff\n\quad \text{authenticationTest inputOK}\n\quad [\text{Name PlatoonSergeant says prop (SOME (SLc (PL pl\text{Command}))))] \land\n\quad \text{CFGInterpret} (M, Oi, Os)\n\quad (\text{CFG inputOK secContext secContextNull}\n\quad ([\text{Name PlatoonSergeant says prop (SOME (SLc (PL pl\text{Command}))))]}::\text{ins} \ s\ outs) \land\n\quad (M, Oi, Os) \text{ sat prop NONE}\n\n[\text{PlatoonSergeant\_trap\_pl\text{Command}\_justified\_thm}]\n\n\vdash \forall NS\ Out\ M\ Oi\ Os.\n\quad \text{TR} (M, Oi, Os) (\text{trap} [\text{SOME (SLc (PL pl\text{Command}))}])\n\quad (\text{CFG inputOK secContext secContextNull}\n\quad ([\text{Name PlatoonSergeant says prop (SOME (SLc (PL pl\text{Command}))))]}::\text{ins} \ s\ outs)\n\quad (\text{CFG inputOK secContext secContextNull} ins\n\quad (NS\ s\ (\text{trap} [\text{SOME (SLc (PL pl\text{Command}))}]))\n\quad (Out\ s\ (\text{trap} [\text{SOME (SLc (PL pl\text{Command}))))]}::\text{outs})) \iff\n\quad \text{authenticationTest inputOK}\n\n161\]
8.4 Discussion

Once the SSMs were described (as in the previous chapter) there were no problems verifying them in HOL. Parametrizing the SSMs made it easier to verify them in HOL. Nevertheless, it was much easier to generate HOL code automatically using the HOL-generator. The by-hand generated HOL proofs for one SSM took several days to a week each to code and debug, even with the parametrizable SSM. Whereas, six SSMs where generated and tested in less than four hours using the HOL-generator.

The next part of this thesis discusses the application of STORM to the patrol base operations. It then discusses improvements that could extend the capabilities of STORM.
Part III

Discussion, Conclusion, and Future Work
Chapter 9

STORM Conformance to SSE Framework

The purpose of this thesis is to demonstrate STORM on an example of a non-automated, human-centered system. This section extends beyond the original goal of this thesis and examines STORM itself.

Conformance to the System Security Engineering Framework described in NIST SP 800-160 requires a closer look. The SSE Frameworks is shown in figure 9.1. This chapter discusses how STORM conforms to the SSE Framework. The Framework has three components each with various activities that the analyst should perform. These are:

- Problem
  - Define Security Objectives
  - Define Security Requirements
  - Define Success Measures
  - Define Life-Cycle Security Concepts
– Produce Evidence for Security Aspects of the Problem

• Solution

– Define the Security Aspects of the Solution
– Realize the Security Aspects of the Solution
– Produce Evidence for Security Aspects of the Solution

• Trustworthiness

– Develop Assurance Case for Acceptable Security
– Demonstrate Assurance Case Is Satisfied

STPA/STPA-Sec focuses primarily on the problem and solution components. CSBD focuses primarily on the Trustworthiness component.

9.1 Problem

Define Security Objectives  STORM conforms to this activity in step 1 of the STPA/STPA-Sec analysis. In this step, the purpose of the analysis is defined. The goal of complete mediation in CSBD also defines security objectives.

Define Security Requirements  STORM conforms to this activity in step 1 of the STPA/STPA-Sec analysis. In this step, the stakeholder needs are defined and the analysis proceeds by defining asset losses and unacceptable asset losses with respect to these needs.

Define Success Measures  STORM conforms to this activity in step 1 of the STPA/STPA-Sec analysis and in CSBD. In step 1 of the STPA/STPA-Sec analysis,
Figure 9.1: Systems security engineering Framework. (Image from NIST Special Publication 800-160 Vol 1: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.[8])

System-level constraints are a measure of success. These are the top-level security goals. They are realized in step 4 and linked to the losses through the system-level hazards. If the system satisfies these constraints then the system or mission analysis is successful. In addition, complete mediation is a requirement of CSBD.

**Define Life Cycle Security Concepts** STORM does not explicitly define life-cycle concepts. That is, it is not explicitly discussed in any of the STPA or CSBD steps. But a thorough analysis would include consideration of life-cycle concepts with regards to worst-case scenarios in step 4. Adding the risk management framework discussed in the next chapter would explicitly account for system-life cycle processes in the analysis.
Produce Evidence for Security Aspects of the Problem  STORM produces evidence for the security aspects of the problem in the STPA/STPA-Sec analysis and CSBD. Evidence is the stakeholder’s definition of assets and asset losses. These are made more and more explicit in the delineation of losses and hazards and vulnerabilities. The structure of the CONOPS developed into an SSM is also evidence of the security aspect of the problem.

9.2 Solution

Define the Security Aspects of the Solution  STORM conforms to this activity in several areas. First, it defines the system-level and refined constraints corresponding to each hazard or vulnerability in step 1 of the analysis. Second, the rationales in step 4 of the analysis also define the security aspects of the solution. Third, complete mediation solutions are defined in the structure of the SSM and the authentication and access-control policy of the CSBD.

Realize the Security Aspects of the Solution  STORM conforms to this activity with regards to complete mediation via CSBD. But STORM only realizes the solution with regards to the concept and design phase and not implementation. The HOL proofs of complete mediation realize the concept and design phase solution for complete mediation. But the rationales determined in step 4 of the STPA/STPA-Sec analysis do not realize the design. To do this, the analysis should include evidence that the rationale is designed into the system. The next chapter discusses the AC Methodology and how components of it can be added to STORM to supplement its documentation. This documentation would also serve to realize more of the security aspects of the solution.

Realization of the security aspects should be followed through in the implementation phase. The implementation phase is a component of the risk management framework.
and is described in the next chapter.

**Produce Evidence for Security Aspects of the Solution**  STORM conforms to this activity with regards to complete mediation. The HOL generated proofs are reproducible, auditable documentation demonstrating complete mediation. What makes this evidence strong is that the HOL proofs are formal methods that use computer-aided reasoning to demonstrate proof of complete mediation.

Nevertheless, evidence for the rationale of the STPA/STPA-Sec analysis could also be documented. The next chapter discusses the AC Methodology and how its use of citing evidence can be added to step 4 of the STPA/STPA-Sec analysis to improve documentation.

Documentation from the implementation phase of the risk management framework would provide evidence of the security aspects of the solution beyond the concept and design phase.

### 9.3 Trustworthiness

**Develop Assurance Case for Acceptable Security**  STORM does this for the CSBD proofs of complete mediation. The assurance case is that the system satisfies the property of complete mediation. It is also implicit in the structure of the SSMs. With regards to the constraints and the rationale in the STPA/STPA-Sec, assurance cases could be built. The AC Methodology and its approach to assurance case development could be added to STORM.

**Demonstrate Assurance Case is Satisfied**  The verification of the CSBD proofs of complete mediation demonstrate that the assurance cases are satisfied. This could also
be done for the constraints and rationale of the STPA/STPA-Sec via the example of the AC Methodology.

STORM does conform to the SSE Framework. But it could be improved to extend it beyond just demonstrating complete mediation.
Chapter 10

Extending STORM to A Comprehensive Assurance Methodology

The DoD and the U.S. government as a whole are demonstrative of the type of complex, multi-component systems that STORM aims to address. The problem of both ineffective and redundant mission assurance practices in the DoD is the driving force behind the 2012 Mission Assurance Strategy written by the U.S. Secretary of Defense [14]. This chapter describes two means to make STORM more comprehensive and more trustworthy.

10.1 STORM on The Risk Management Framework

Note that the STPA/STPA-Sec analysis focuses on hazards or vulnerabilities. But from the perspective of decision makers, these are more generally categorized as risks. Risks are anything that can potentially cause a loss to the stakeholder. This includes other
types of risk such as privacy and financial risks. There is an STPA analysis for privacy called STPA-Priv. This could be integrated into STORM. But financial and other types of risk should be assessed in another part of the system development process. The exception is the financial impact to the stakeholders of hazard, safety, and possibly privacy risks covered in the STORM analysis.

Steps 1 through 4 of STPA/STPA-Sec identify risks. Step 4 also generates risk-reducing solutions. But decision makers need to be informed of the potential impact to the stakeholder of each risk. Therefore, what is missing is a methodology for prioritizing risks. What follows is a discussion of how this could be done based on the Risk Management Framework (NIST SP 800-53).

10.1.1 Risk Management Framework (NIST SP 800-37)

The Risk Management Framework is discussed in NIST SP 800-37. It is composed of six components.

- Categorize Risks (vulnerabilities)
- Select Risk (security) Controls
- Implement Controls
- Assess
- Monitor
- Authorize
10.1.1.1 Risk Categories

Throughout each phase of the STPA/STPA-Sec analysis, the potential impact of a discovered risk should be assessed based on predefined categories. These category definitions should be defined by the stakeholders. The best practice would be to use categories that are informed by industry standards. For example, NIST SP 800-60, Revision 1, Guide for Mapping Types of Information and Information Systems to Security Categories, (Volumes 1 and 2) provides category guidelines for Federal information systems. Such practices highlight conformance to industry norms and profit from the wisdom of experts.

Category definitions should be defined by the stakeholders and passed to the analysts. But if they are not, then defining these categories should be an early step in the STORM analysis. One approach would be to add a third component of STORM that precedes the STPA/STPA-Sec analysis. Another means of accomplishing this would be to add it to Step 1 of the STPA/STPA-Sec analysis.

A risk is assigned a risk label based on an appropriately selected category. The risk categories categorize types of risks based on how they effect the organization. Each category may impose some level of acceptable security for managing the risk. For example, risks may be low, moderate, or high or they could be low, injurious, or fatal. Higher risks may require more stringent safety and security controls. For example, higher risks may require consultation with a lawyer or presidential authorization.

Although not demonstrated in this thesis, CSBD also manages risk categories. It does so by using security and integrity levels. Security levels, for example, can be defined as unclassified, classified, and top secret. Principals are granted access to read or modify information based on their assigned risk level and the risk level of the object they wish to access. In the SSMs that were verified for the patrol base operations, the assignment of commands to principals can also be thought of as assigning risk labels.
The hazards, vulnerabilities, and unsafe control actions in the STPA/STPA-Sec could also be assigned risk labels. Safety and security controls should then reflect the level of risk.

The risk labels also help analysts prioritize risks. This is particularly useful for missions which require rapid development of CONOPS because the higher-level risk actions will take center stage. For larger projects with more lengthy analysis time, risk labels help decision makers calculate and decide which risks are acceptable and which are not.

### 10.1.1.2 Control Selection

After risks are categorized, decision makers must select which risk-mitigating controls to implement. The categories guide the selection. The industry, stakeholders, and mission also guide the selection.

CSBD selects controls by the access-control policy, that is the authentication and authorization functions.

STPA/STPA-Sec should select security controls based on the type of mission. For example, if the type of mission could result in loss of life, then risks that could result in loss of life should be labeled as such. Controls for such risks may include bringing extra survival gear to avoid loss of life, performing extra drills to avoid loss of life, organizing for additional rescue support to be available during the mission, and so on.

Selecting controls should be integrated into the constraints section of step 1 and the rationale section of step 4 of the STPA/STPA-Sec analysis.

With the risk categories predefined and risk labels assigned to risks, selecting controls is faster because it narrows the choices. It is also potentially more reliable because each control already has well-defined actions. These actions will likely be peer-reviewed and
optimized. They could be readily adapted to fit the mission.

Before moving to the next activity CSBD should verify the CONOPS on the mission.

10.1.1.3 Implementation

Implementation refers to finalizing the CONOPS.

10.1.1.4 Assess

An assessment strategy should reflect stakeholder needs and compliance requirements and should be informed by the STPA/STPA-Sec and CSBD analysis. If the CONOPS are assessed and deemed acceptable, then they will be handed to the decision makers for authorization. Otherwise, improvements will be suggested and analysis revisited.

10.1.1.5 Authorize

Authorization to make the system operational should be informed by the analysis, risk calculations, and assessment strategy. If the risks are acceptable, then the system or mission should be made operational. Otherwise, that analysis should be revisited, rationale updated, and controls re-selected to bring the risk to an acceptable level.

10.1.1.6 Monitor

A monitoring strategy should be designed during the analysis to set performance goals and acceptable measures of success. These should be defined in step 1 of the STPA/STPA-Sec analysis. Monitoring the system or mission validates the CONOPS. It
also provides insight for future missions and demonstrates that stakeholder needs were met.

Explicitly adding the six risk steps of the risk management framework would add another level of NIST conformance to STORM. It would make it better by making it more applicable to missions and systems that require conformance to the Risk Management Framework. It would improve its range of applicability. It would also make STORM more effective.

The next section discusses an addition to STORM that would extend the documentation and thus demonstration of trustworthiness.

10.2 STORM Documentation Upgrade: The Assurance Case Methodology

The need for trustworthy systems is driving current research endeavors in all areas of systems and mission assurance analysis. This is also true of computer architecture. In "Hardware Security Evaluation Using Assurance Case Models" published in "2015 10th International Conference on Availability, Reliability, and Security," Kawakami, et. al. describe a methodology for designing assured computer architectures [1]. The methodology lacks in some areas where STORM excels. But it does have an advantage in its documentation practices that could enhance STORM. A comparison is made of the two methodologies. Lessons learned from the AC methodology are suggested as an improvements to STORM.
10.2.1 STORM Compared to The AC Methodology

A comparison is helpful to understanding the pros and cons of each methodology.

The AC methodology also presents a four step process. The steps are not clearly stated in the paper. This is a best interpretation of the steps.

- **Step 1**: State the global desired system security property.
- **Step 2**: Expand the goal from the previous step. (This step breaks the goal into smaller steps.)
- **Step 3**: Show how strategies derive the subgoals.
- **Step 4**: List evidence that each goal is satisfied or where the goal needs to be satisfied.

Recall again the four steps from STORM’s STPA-Sec analysis (repeated here for ease of reading).

- **Step 1**: Define the purpose of the analysis.
- **Step 2**: Model the control structure.
- **Step 3**: Identify unsafe control actions.
- **Step 4**: Identify loss scenarios.

**Goals Must Match Stakeholder Needs** The AC methodology does not make the stakeholder needs explicit in the analysis. It describes a goal-by-goal process. Goals are determined based on the analyst’s experience. On the other hand, the STPA/STPA-Sec analysis of STORM describes a system-level process. It begins at the top-level of the
system and derives the goals explicitly based on stakeholder needs and the concept of assets and asset loss minimization. To put this in more down to earth terms, I may have achieved the goal of preventing the ship from sinking. But, what happens if I am on a plane?

A Philosophy for How Hazards And Vulnerabilities Occur Is Necessary to Identify Them The AC methodology provides no guidance for how to identify hazards and vulnerabilities. It lacks a clear philosophy of what causes them. STPA/STPA-Sec, on the other hand, is founded on the STAMP philosophy that hazards and vulnerabilities are caused by unsafe control actions in addition to process model failures and component failures. This philosophy is founded on years of research by MIT Professor Nancy Leveson who is an expert in accident analysis.

A System-level Model Is Necessary to Identify Hazards and Vulnerabilities The AC Methodology provides no system-level model. But a model is necessary to envision how parts of the system interact and to search for hazards and vulnerabilities. STPA does this. Step 2 diagrams a functional control model that is based on the STAMP philosophy. This model focuses on the component processes (component-level analyses) and process interactions (system-level analyses).

The Analysis Must Be Thorough And Consider All Potential Hazards And Vulnerabilities The AC Methodology identifies hazards and vulnerabilities based on goals that are derived from the analyst’s experiences. But any large-scale analysis is multi-faceted and complex. Humans in all their greatness still require guidance to ensure thoroughness. STPA/STPA-Sec delineates a comprehensive set of categories of unsafe control actions. This guides the analyst to think about each type of action and how it could cause a hazard or vulnerability in the worse-case scenario. Furthermore, the Thomas method systematically addresses all combinations of system state and
action that could cause an UCA.

**Documentation Is Important!** What the AC Methodology does and does well is document that each assurance case is satisfied. It does so by stating explicitly a goal and its subgoals and then documents evidence that the goal is met (the assurance case is satisfied) or that the goal requires attention (assurance case needs to be satisfied). STORM also documents the goal and subgoals in steps 1 through 4. But it does not explicitly state evidence that a subgoal is met (a rationale implemented) or needs to be met. CSBD does this, but only for the security triad confidentiality, integrity, and availability (CIA).

Adding this evidence requirement to the STPA/STPA-Sec component of STORM would complete its documentation. This could be done by adding another column in the step 4 tables. This column would document evidence wherein the rationale is satisfied or not and possibly assign responsibility to an analyst.

**Demonstrating Compliance And Stakeholder Needs Are Met Is Important**

What the AC Methodology also does well is demonstrate compliance easily using a goal tree structure, which maps the goal from the top-level through each subgoal and to the evidence. It is organized and easy to see that a goal is satisfied. This is often necessary to demonstrate explicitly that the system satisfies stakeholder needs or complies with regulations. The STPA/STPA-Sec analysis of STORM satisfies the goal and each goal can be traced through the top-level analysis through to the subgoals. But the direct link connecting each top goal to evidence of rationale is not documented explicitly. It requires the analyst to sort through the many tables and pick out the specific subgoals and rationale that satisfy each goal.

Generating a goal tree in the STPA/STPA-Sec for each top-level goal would explicitly demonstrate that stakeholder needs are met and that the system complies with
regulations. This would be straightforward if the STPA/STPA-Sec analysis were automated because the goal (losses or system-level constraints) could be selected where the automation process would select all subgoals (unsafe control actions) and rationale/evidence and generate a goal tree.

Otherwise, the analyst should explicitly draw-out the goal tree to document that the system requirements are met. This could be done in a limited capacity. For example, goal trees could be drawn for high level risks, risks with stringent compliance regulations, or for stakeholder needs that are of greatest value.

These two documentation activities would solidify the overall trustworthiness of the system or mission by developing and demonstrating the assurance cases. It would also help analysts see the connections between the goal and the solution, and generate greater confidence in the CONOPS. It may even shine light on weaknesses of the analysis. If rationale were assigned to individuals, these two activities would aid in accountability. The system or mission would also be easier to audit because assurance cases would be explicit and easy to read.¹

STORM is an effective methodology for developing CONOPS that meet stakeholder needs and conform to the SSE Framework. But it could be made better. The suggestions here would extend STORM to a more comprehensive system and mission assurance methodology by bringing it into compliance with the NIST SP 800-37 risk management framework and strengthening the trustworthiness component of the SSE Framework.

The next chapter discusses future work on the patrol base operations and STORM.

¹Making life easier for the auditors keeps them happy.
Chapter 11

Future Work

This chapter suggests some ideas for future work on the material discussed in this thesis.

11.1 Patrol Base Operations

Any thorough analysis of any system or mission requires not only someone who understands the analysis methodology but also someone who understands the system or mission. This thesis began as a demonstration of STORM on a non-automated, human-centered system. The CSBD was partially worked through with a subject matter expert in military operations. The STPA analysis was not. It would be useful to work through the analysis rigorously with an expert and a purpose (i.e., to identify mission improvements) and to extend the analysis through to develop new CONOPS. It is likely that CONOPS in the mission could be improved, as hinted at by the subject matter expert on this project. However, at this time, no such analysis is planned.
11.2 CSBD on Patrol Base Operations

At the time of this thesis CSBD has three parametrizable SSMs. The simplest one was demonstrated in this thesis. The other two add a vertical refinement to the model and are discussed next.

11.2.0.1 Authentication

Authentication for our model of the patrol base operations is assumed through visual recognition of an authority. This was done to simplify the analysis. But CSBD has a parametrizable SSM that allows for a cryptographic authentication function. This function requires that transitions are executed if and only if symmetric or asymmetric keys are verified. Such keys could be used on future models of the patrol base operations.

11.2.0.2 Roles

Authentication for our model of the patrol base operations also assumed authorities with names such as Platoon Leader and Platoon Sergeant. But these are roles and it is the actual people acting in these roles that are issuing commands (i.e., Captain Jane Smith acting as Platoon Leader). A parametrized version of an SSM which accounts for people acting in roles also exits and could be used on future models of the patrol base operations.

11.3 STORM

STORM could be made more user friendly through automation.
11.3.1 Automation

Pen and paper analysis are cumbersome, especially for large projects. They are also difficult to collaborate on and to disseminate. But automation solves that task. A STORM user application would facilitate analysis by queuing analysts for specific information, verifying assurance proofs, and generating documentation. Furthermore, all the data would be contained in a database that could be easily searched by humans or artificial intelligence algorithms and easily shared on the cloud with other analysts, stakeholders, and auditors.

Efforts to automate STPA/STPA-Sec and CSBD have already been undertaken. An automated version of STPA, STPA-Sec, and STPA-Priv (for privacy) were developed by Asim Abdulkhaleq and Stefan Wagner [19]. Their tool called XSTAMPP allowed the user to generate lists for losses and vulnerabilities, draw a functional model of the system, describe unsafe control actions, and input rationale. Parts of the analysis were automated.

CSBD has been partially automated. Fully automated theorem proving remains a hard problem in computer science. However, the parametrization of secure state machines in HOL makes automation for this subset of theorem proving problems straight forward. Some of the proofs in this thesis were verified with HOL script files automatically generated in Haskell.

With the components of STORM built into a computer application, analysts could open up an app on their computer or iPad, draw their system, functional control model, and SSM. They would be prompted by a form that asks for information about stakeholders, assets, losses, hazards, vulnerabilities, and unsafe control actions. Additional prompts would ask for rationale on how to mitigate the problem. The form could also ask for evidence to demonstrate the problem is solved or assign such a task to other team members. In addition, a form would require information about access-control policies
and controllers. The application would then link information to the diagrams and generate proofs of complete mediation. Upon request, a variety of forms could be generated to demonstrate compliance or to disseminate the CONOPS to leaders and team members.

The information could be saved in a database on the web which could be searched and disseminated. Anyone with an internet connection and access rights could access the information, allowing for cross-organizational collaboration and thus greater visibility. Such an application would eliminate redundancy because analysts could search the cloud for previously analyzed or developed CONOPS and reuse or modify and reuse them. Furthermore, the application could automatically generate requests for mission approval and resources. Any number of things could be done once the system is automated. This would save time, resources, and effort and lead to more accurate reporting.

Automating STORM and adding components of the RMF and AC Methodology would make STORM a more comprehensive and analyst friendly system and mission assurance methodology that would generate trustworthy CONOPS the satisfy stakeholder needs.
Appendices
Appendix A

Access Control Logic Theories: Pretty-Printed Theories
Contents

1 aclfoundation Theory 3
  1.1 Datatypes ........................................... 3
  1.2 Definitions ........................................... 3
  1.3 Theorems ............................................ 4

2 aclsemantics Theory 6
  2.1 Definitions ........................................... 6
  2.2 Theorems ............................................ 8

3 aclrules Theory 10
  3.1 Definitions ......................................... 11
  3.2 Theorems ........................................... 11

4 aclDrules Theory 17
  4.1 Theorems ........................................... 17
1 aclfoundation Theory

Built: 25 February 2018

Parent Theories: indexedLists, patternMatches

1.1 Datatypes

Form =
    TT
    | FF
    | prop 'aavar
    | notf ('aavar, 'apn, 'il, 'sl) Form)
    | (andf) ('aavar, 'apn, 'il, 'sl) Form
    | (orf) ('aavar, 'apn, 'il, 'sl) Form)
    | (impf) ('aavar, 'apn, 'il, 'sl) Form)
    | (eqf) ('aavar, 'apn, 'il, 'sl) Form)
    | (says) ('apn Princ) ('aavar, 'apn, 'il, 'sl) Form)
    | (speaks_for) ('apn Princ) ('apn Princ)
    | (controls) ('apn Princ) ('aavar, 'apn, 'il, 'sl) Form)
    | reps ('apn Princ) ('apn Princ)
    | (domi) ('apn, 'il) IntLevel)
    | (eqi) ('apn, 'il) IntLevel)
    | (doms) ('apn, 'sl) SecLevel)
    | (eqs) ('apn, 'sl) SecLevel)
    | (eqn) num num
    | (lte) num num
    | (lt) num num

Kripke =
    KS ('aavar -> 'aaworld -> bool)
    ('apn -> 'aaworld -> 'aaworld -> bool) ('apn -> 'il)
    ('apn -> 'sl)

Princ =
    Name 'apn
    | (meet) ('apn Princ) ('apn Princ)
    | (quoting) ('apn Princ) ('apn Princ)

IntLevel = iLab 'il | il 'apn

SecLevel = sLab 'sl | sl 'apn

1.2 Definitions
[imapKS_def]
\[ \vdash \forall \text{Intp Jfn ilmap slmap}. \]
\[ \text{imapKS (KS Intp Jfn ilmap slmap)} = \text{ilmap} \]

[intpKS_def]
\[ \vdash \forall \text{Intp Jfn ilmap slmap}. \]
\[ \text{intpKS (KS Intp Jfn ilmap slmap)} = \text{Intp} \]

[jKS_def]
\[ \vdash \forall \text{Intp Jfn ilmap slmap}. \]
\[ \text{jKS (KS Intp Jfn ilmap slmap)} = \text{Jfn} \]

[01_def]
\[ \vdash 01 = \text{PO one_weakorder} \]

[one_weakorder_def]
\[ \forall x y. \text{one_weakorder x y } \iff T \]

[po_TY_DEF]
\[ \exists \text{rep. TYPE_DEFINITION WeakOrder rep} \]

[po_tybij]
\[ \vdash (\forall a. \text{PO (repPO a)} = a) \land \]
\[ \forall r. \text{WeakOrder r } \iff (\text{repPO (PO r)} = r) \]

[prod_PO_def]
\[ \vdash \forall \text{PO}_1 \text{PO}_2. \]
\[ \text{prod_PO \text{PO}_1 \text{PO}_2 = \text{PO (RPROD (repPO \text{PO}_1) (repPO \text{PO}_2))}} \]

[smapKS_def]
\[ \vdash \forall \text{Intp Jfn ilmap slmap}. \]
\[ \text{smapKS (KS Intp Jfn ilmap slmap)} = \text{slmap} \]

[Subset_PO_def]
\[ \vdash \text{Subset_Po = PO (\subseteq)} \]

### 1.3 Theorems

[abs_po11]
\[ \vdash \forall r r'. \]
\[ \text{WeakOrder r } \Rightarrow \text{WeakOrder r' } \Rightarrow ((\text{PO r = PO r'}) \iff (r = r')) \]

[absPO_fn_onto]
\[ \vdash \forall a. \exists r. (a = \text{PO r}) \land \text{WeakOrder r} \]
[antisym_prod_antisym]
\(\vdash \forall r. s.\)
\(\text{antisymmetric } r \land \text{antisymmetric } s \Rightarrow \text{antisymmetric } (\text{RPROD } r \ s)\)

[EQ_WeakOrder]
\(\vdash \text{WeakOrder } (=)\)

[KS_bij]
\(\vdash \forall M. \ M = \text{KS } (\text{intpKS } M) (\text{jKS } M) (\text{imapKS } M) (\text{smapKS } M)\)

[one_weakorder_WO]
\(\vdash \text{WeakOrder } \text{one_weakorder}\)

[onto_po]
\(\vdash \forall r. \text{WeakOrder } r \iff \exists a. r = \text{repPO } a\)

[po_bij]
\(\vdash (\forall a. \text{PO } (\text{repPO } a) = a) \land \forall r. \text{WeakOrder } r \iff (\text{repPO } (\text{PO } r) = r)\)

[PO_repPO]
\(\vdash \forall a. \text{PO } (\text{repPO } a) = a\)

[refl_prod_refl]
\(\vdash \forall r. s. \text{reflexive } r \land \text{reflexive } s \Rightarrow \text{reflexive } (\text{RPROD } r \ s)\)

[repPO_iPO_partial_order]
\(\vdash (\forall x. \text{repPO } iPO x x) \land (\forall x y. \text{repPO } iPO x y \land \text{repPO } iPO y x \Rightarrow (x = y)) \land (\forall x y z. \text{repPO } iPO x y \land \text{repPO } iPO y z \Rightarrow \text{repPO } iPO x z)\)

[repPO_01]
\(\vdash \text{repPO } 01 = \text{one_weakorder}\)

[repPO_prod_PO]
\(\vdash \forall po_1 po_2.\)
\(\text{repPO } (\text{prod_PO } po_1 po_2) = \text{RPROD } (\text{repPO } po_1) (\text{repPO } po_2)\)

[repPO_Subset_PO]
\(\vdash \text{repPO } \text{Subset_PO} = (\subseteq)\)

[RPROD_THM]
\(\vdash \forall r s a b.\)
\(\text{RPROD } r \ s \ a \ b \iff r \ (\text{FST } a) \ (\text{FST } b) \land s \ (\text{SND } a) \ (\text{SND } b)\)
ACLSEMANTICS THEORY

[SUBSET_WO]
\[\text{WeakOrder} (\subseteq)\]

[trans_prod_trans]
\[\forall r s. \text{transitive } r \land \text{transitive } s \Rightarrow \text{transitive} \ (\text{RPROD } r \ s)\]

[WeakOrder_Exists]
\[\exists R. \text{WeakOrder } R\]

[WO_prod_WO]
\[\forall r s. \text{WeakOrder } r \land \text{WeakOrder } s \Rightarrow \text{WeakOrder} \ (\text{RPROD } r \ s)\]

[WO_repPO]
\[\forall r. \text{WeakOrder } r \iff (\text{repPO} (\text{PO } r) = r)\]

2 aclsemantics Theory

Built: 25 February 2018
Parent Theories: aclfoundation

2.1 Definitions

[Efn_def]
\[\forall Oi Os M. \text{Efn } Oi Os M \ TT = \mathcal{U}(\cdot') \land\]
\[(\forall Oi Os M. \text{Efn } Oi Os M \ FF = \{\} \land\]
\[(\forall Oi Os M \ p. \text{Efn } Oi Os M \ (\text{prop } p) = \text{intpKS } M \ p) \land\]
\[(\forall Oi Os M \ f. \]
\[\text{Efn } Oi Os M \ (\text{notf } f) = \mathcal{U}(\cdot') \text{DIFF } \text{Efn } Oi Os M \ f) \land\]
\[(\forall Oi Os M \ f_1 f_2. \]
\[\text{Efn } Oi Os M \ (f_1 \text{ andf } f_2) =\]
\[\text{Efn } Oi Os M \ f_1 \cap \text{Efn } Oi Os M \ f_2) \land\]
\[(\forall Oi Os M \ f_1 f_2. \]
\[\text{Efn } Oi Os M \ (f_1 \text{ orf } f_2) =\]
\[\text{Efn } Oi Os M \ f_1 \cup \text{Efn } Oi Os M \ f_2) \land\]
\[(\forall Oi Os M \ f_1 f_2. \]
\[\text{Efn } Oi Os M \ (f_1 \text{ impf } f_2) =\]
\[\mathcal{U}(\cdot') \text{DIFF } \text{Efn } Oi Os M \ f_1 \cup \text{Efn } Oi Os M \ f_2) \land\]
\[(\forall Oi Os M \ f_1 f_2. \]
\[\text{Efn } Oi Os M \ (f_1 \text{ eqf } f_2) =\]
\[\mathcal{U}(\cdot') \text{DIFF } \text{Efn } Oi Os M \ f_1 \cup \text{Efn } Oi Os M \ f_2) \land\]
\[(\forall Oi Os M \ (P \text{ says } f) =\]
\[\{w \mid \text{Jext } (\text{jKS } M) \ P \ w \subseteq \text{Efn } Oi Os M \ f\}) \land\]
\[(\forall Oi Os M \ P \ Q. \]
\[\text{Efn } Oi Os M \ (P \text{ speaks_for } Q) =\]
if \text{Jext} (\text{jKS} M) Q \text{RSUBSET} \text{Jext} (\text{jKS} M) P \text{then} \mathcal{U}(\cdot:\nu)
else \{\} \land 
(\forall \text{O}i \text{ Os} M P j.
\text{Efn} \text{ Oi} \text{ Os} M (P \text{ controls } f) = 
\mathcal{U}(\cdot:\nu) \text{ DIFF} \{w \mid \text{Jext} (\text{jKS} M) P w \subseteq \text{Efn} \text{ Oi} \text{ Os} M f\} \cup
\text{Efn} \text{ Oi} \text{ Os} M f\} \land
(\forall \text{O}i \text{ Os} M P Q j.
\text{Efn} \text{ Oi} \text{ Os} M (\text{reps} P Q f) = 
\mathcal{U}(\cdot:\nu) \text{ DIFF} \{w \mid \text{Jext} (\text{jKS} M) (P \text{ quoting } Q) w \subseteq \text{Efn} \text{ Oi} \text{ Os} M f\} \cup
\{w \mid \text{Jext} (\text{jKS} M) Q w \subseteq \text{Efn} \text{ Oi} \text{ Os} M f\}\} \land
(\forall \text{O}i \text{ Os} M \text{ int}_1 \text{ int}_2. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{int}_1 \text{ dom} \text{ int}_2) = 
\text{if} \text{ repPO} \text{ Oi} (\text{Lifn} M \text{ int}_2) (\text{Lifn} M \text{ int}_1) \text{ then} \mathcal{U}(\cdot:\nu)
else \{\} \land
(\forall \text{O}i \text{ Os} M \text{ int}_2 \text{ int}_1. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{int}_2 \text{ eq} \text{ int}_1) = 
(\text{if} \text{ repPO} \text{ Oi} (\text{Lifn} M \text{ int}_2) (\text{Lifn} M \text{ int}_1) \text{ then} \mathcal{U}(\cdot:\nu)
else \{\} \land
\text{if} \text{ repPO} \text{ Oi} (\text{Lifn} M \text{ int}_1) (\text{Lifn} M \text{ int}_2) \text{ then} \mathcal{U}(\cdot:\nu)
else \} \land
(\forall \text{O}i \text{ Os} M \text{ secl}_1 \text{ secl}_2. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{secl}_1 \text{ dom} \text{ secl}_2) = 
\text{if} \text{ repPO} \text{ Oi} (\text{Lsfn} M \text{ secl}_2) (\text{Lsfn} M \text{ secl}_1) \text{ then} \mathcal{U}(\cdot:\nu)
else \} \land
(\forall \text{O}i \text{ Os} M \text{ secl}_2 \text{ secl}_1. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{secl}_2 \text{ eq} \text{ secl}_1) = 
(\text{if} \text{ repPO} \text{ Oi} (\text{Lsfn} M \text{ secl}_2) (\text{Lsfn} M \text{ secl}_1) \text{ then} \mathcal{U}(\cdot:\nu)
else \{\} \land
\text{if} \text{ repPO} \text{ Oi} (\text{Lsfn} M \text{ secl}_1) (\text{Lsfn} M \text{ secl}_2) \text{ then} \mathcal{U}(\cdot:\nu)
else \} \land
(\forall \text{O}i \text{ Os} M \text{ numExp}_1 \text{ numExp}_2. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{numExp}_1 \text{ eqn} \text{ numExp}_2) = 
\text{if} \text{ numExp}_1 = \text{ numExp}_2 \text{ then} \mathcal{U}(\cdot:\nu) \text{ else } \{\} \land
(\forall \text{O}i \text{ Os} M \text{ numExp}_1 \text{ numExp}_2. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{numExp}_1 \text{ lte} \text{ numExp}_2) = 
\text{if} \text{ numExp}_1 \leq \text{ numExp}_2 \text{ then} \mathcal{U}(\cdot:\nu) \text{ else } \{\} \land
\forall \text{O}i \text{ Os} M \text{ numExp}_1 \text{ numExp}_2. 
\text{Efn} \text{ Oi} \text{ Os} M (\text{numExp}_1 \text{ lt} \text{ numExp}_2) = 
\text{if} \text{ numExp}_1 < \text{ numExp}_2 \text{ then} \mathcal{U}(\cdot:\nu) \text{ else } \{\}

[\text{Jext\_def}]
\vdash (\forall J s. \text{Jext} J (\text{Name} s) = J s) \land
(\forall J P_1 P_2.
\text{Jext} J (P_1 \text{ meet} P_2) = \text{Jext} J P_1 \text{ RUNION} \text{Jext} J P_2) \land
\forall J P_1 P_2. \text{Jext} J (P_1 \text{ quoting} P_2) = \text{Jext} J P_2 \circ \text{Jext} J P_1

[\text{Lifn\_def}]
\vdash (\forall M l. \text{Lifn} M (\text{iLab} l) = l) \land
\forall M \text{name}. \text{Lifn} M (\text{il name}) = \text{imapKS} M \text{name}
\[Lsfn\_def\]
\[∀ M \ l. \ Lsfn \ M \ (sLab \ l) = l \land \forall M \ name. \ Lsfn \ M \ (sl \ name) = smapKS \ M \ name\]

### 2.2 Theorems

\[\text{andf\_def}\]
\[∀Oi Os M f_1 f_2. \ Efn Oi Os M (f_1 \ andf \ f_2) = Efn Oi Os M f_1 \cap Efn Oi Os M f_2\]

\[\text{controls\_def}\]
\[∀Oi Os M P f. \ Efn Oi Os M (P controls f) = \mathcal{U}(\cdot'v) \ DIFF \{w \mid \text{jext (jKS } M) P w ⊆ Efn Oi Os M f\} \cup Efn Oi Os M f\]

\[\text{controls\_says}\]
\[∀M P f. \ Efn Oi Os M (P controls f) = Efn Oi Os M (P says f \ impf f)\]

\[\text{domi\_def}\]
\[∀Oi Os M intl_1 intl_2. \ Efn Oi Os M (intl_1 domi intl_2) = \text{if repPO } Oi (Lifn M intl_2) (Lifn M Intl_1) \text{ then } \mathcal{U}(\cdot'v) \text{ else } \{\}\]

\[\text{doms\_def}\]
\[∀Oi Os M secl_1 secl_2. \ Efn Oi Os M (secl_1 doms secl_2) = \text{if repPO } Os (Lsfn M secl_2) (Lsfn M secl_1) \text{ then } \mathcal{U}(\cdot'v) \text{ else } \{\}\]

\[\text{eqf\_def}\]
\[∀Oi Os M f_1 f_2. \ Efn Oi Os M (f_1 eqf f_2) = (\mathcal{U}(\cdot'v) \ DIFF \ Efn Oi Os M f_1 \cup Efn Oi Os M f_2) \cap (\mathcal{U}(\cdot'v) \ DIFF \ Efn Oi Os M f_2 \cup Efn Oi Os M f_1)\]

\[eqf\_impf\]
\[∀M f_1 f_2. \ Efn Oi Os M (f_1 eqf f_2) = Efn Oi Os M ((f_1 impf f_2) andf (f_2 impf f_1))\]
Theorems

[eqi_def]
\[ \forall Oi Os M \text{ intL}_1 \text{ intL}_2. \]
\[ \text{Efn Oi Os M (intL}_1 \text{ eqi intL}_2) =\]
\[ (\text{if repPO Oi (Lsfn M intL}_2) (Lsfn M intL}_1) \text{ then } U(:v)\]
\[ \text{ else } \{\}\) ∩\]
\[ (\text{if repPO Oi (Lsfn M intL}_1) (Lsfn M intL}_2) \text{ then } U(:v)\]
\[ \text{ else } \{\}\]

[eqi_domi]
\[ \forall M \text{ intL}_1 \text{ intL}_2. \]
\[ \text{Efn Oi Os M (intL}_1 \text{ domi intL}_2) =\]
\[ \text{Efn Oi Os M (intL}_2 \text{ domi intL}_1 \text{ andf intL}_1 \text{ domi intL}_2)\]

[eqn_def]
\[ \forall Oi Os M \text{ numExp}_1 \text{ numExp}_2. \]
\[ \text{Efn Oi Os M (numExp}_1 \text{ eqn numExp}_2) =\]
\[ (\text{if numExp}_1 = \text{ numExp}_2 \text{ then } U(:v) \text{ else } \{\})\]

[eqs_def]
\[ \forall Oi Os M \text{ secl}_1 \text{ secl}_2. \]
\[ \text{Efn Oi Os M (secl}_1 \text{ eqs secl}_2) =\]
\[ (\text{if repPO Os (Lsfn M secl}_2) (Lsfn M secl}_1) \text{ then } U(:v)\]
\[ \text{ else } \{\}\) ∩\]
\[ (\text{if repPO Os (Lsfn M secl}_1) (Lsfn M secl}_2) \text{ then } U(:v)\]
\[ \text{ else } \{\}\]

[eqs_doms]
\[ \forall M \text{ secl}_1 \text{ secl}_2. \]
\[ \text{Efn Oi Os M (secl}_1 \text{ doms secl}_2) =\]
\[ \text{Efn Oi Os M (secl}_2 \text{ doms secl}_1 \text{ andf secl}_1 \text{ doms secl}_2)\]

[FF_def]
\[ \forall Oi Os M. \text{ Efn Oi Os M FF = } \{\}\]

[impf_def]
\[ \forall Oi Os M f_1 f_2. \]
\[ \text{Efn Oi Os M (f}_1 \text{ impf f}_2) =\]
\[ U(:v) \text{ DIFF Efn Oi Os M f}_1 \cup \text{ Efn Oi Os M f}_2\]

[lt_def]
\[ \forall Oi Os M \text{ numExp}_1 \text{ numExp}_2. \]
\[ \text{Efn Oi Os M (numExp}_1 \text{ lt numExp}_2) =\]
\[ (\text{if numExp}_1 < \text{ numExp}_2 \text{ then } U(:v) \text{ else } \{\})\]

[lte_def]
\[ \forall Oi Os M \text{ numExp}_1 \text{ numExp}_2. \]
\[ \text{Efn Oi Os M (numExp}_1 \text{ lte numExp}_2) =\]
\[ (\text{if numExp}_1 \leq \text{ numExp}_2 \text{ then } U(:v) \text{ else } \{\})\]
3 aclrules Theory

Built: 25 February 2018
Parent Theories: aclsemantics
3.1 Definitions

[sat_def]
\[ \forall M \ Oi Os f. (M,Oi,Os) \text{ sat } f \iff (\exists fn \ Oi Os M \ f = \mathcal{U}(\text{world})) \]

3.2 Theorems

[And_Says]
\[ \forall M \ Oi Os P Q f. \ (M,Oi,Os) \text{ sat } P \text{ meet } Q \text{ says } f \iff \text{eqf } P \text{ says } f \text{ andf } Q \text{ says } f \]

[And_Says_Eq]
\[ \forall M \ Oi Os P Q f. \ (M,Oi,Os) \text{ sat } P \text{ meet } Q \text{ says } f \iff \text{eqf } P \text{ says } f \text{ andf } Q \text{ says } f \]

[and_says_lemma]
\[ \forall M \ Oi Os P Q f. \ (M,Oi,Os) \text{ sat } P \text{ meet } Q \text{ says } f \iff \text{impf } P \text{ says } f \text{ andf } Q \text{ says } f \]

[Controls_Eq]
\[ \forall M \ Oi Os P f. \ (M,Oi,Os) \text{ sat } P \text{ controls } f \iff (M,Oi,Os) \text{ sat } P \text{ says } f \text{ impf } f \]

[DIFF_UNIV_SUBSET]
\[ \forall M \ Oi Os P Q f. \ (M,Oi,Os) \text{ sat } P \text{ meet } Q \text{ says } f \iff (M,Oi,Os) \text{ sat } P \text{ says } f \text{ impf } f \]

[domi_antisymmetric]
\[ \forall M \ Oi Os l_1 l_2. \ (M,Oi,Os) \text{ sat } l_1 \text{ domi } l_2 \Rightarrow \ (M,Oi,Os) \text{ sat } l_2 \text{ domi } l_1 \Rightarrow \ (M,Oi,Os) \text{ sat } l_1 \text{ eqi } l_2 \]

[domi_reflexive]
\[ \forall M \ Oi Os l. \ (M,Oi,Os) \text{ sat } l \text{ domi } l \]

[domi_transitive]
\[ \forall M \ Oi Os l_1 l_2 l_3. \ (M,Oi,Os) \text{ sat } l_1 \text{ domi } l_2 \Rightarrow \ (M,Oi,Os) \text{ sat } l_2 \text{ domi } l_3 \Rightarrow \ (M,Oi,Os) \text{ sat } l_1 \text{ domi } l_3 \]

[doms_antisymmetric]
\[ \forall M \ Oi Os l_1 l_2. \ (M,Oi,Os) \text{ sat } l_1 \text{ doms } l_2 \Rightarrow \ (M,Oi,Os) \text{ sat } l_2 \text{ doms } l_1 \Rightarrow \ (M,Oi,Os) \text{ sat } l_1 \text{ eqs } l_2 \]
[doms_reflexive] 
\( \vdash \forall M \ Oi \ Os \ l. \ (M, Oi, Os) \ \text{sat} \ l \ \text{doms} \ l \)

[doms_transitive] 
\( \vdash \forall M \ Oi \ Os \ l_1 \ l_2 \ l_3. \\
(M, Oi, Os) \ \text{sat} \ l_1 \ \text{doms} \ l_2 \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ l_2 \ \text{doms} \ l_3 \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ l_1 \ \text{doms} \ l_3 \)

[eqf_and_impf] 
\( \vdash \forall M \ Oi \ Os \ f_1 \ f_2. \\
(M, Oi, Os) \ \text{sat} \ f_1 \ \text{eqf} \ f_2 \ \iff \\
(M, Oi, Os) \ \text{sat} \ (f_1 \ \text{impf} \ f_2) \ \text{andf} \ (f_2 \ \text{impf} \ f_1) \)

[eqf_andf1] 
\( \vdash \forall M \ Oi \ Os \ f \ f' \ g. \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ f' \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ f \ \text{andf} \ g \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ f' \ \text{andf} \ g \)

[eqf_andf2] 
\( \vdash \forall M \ Oi \ Os \ f \ f' \ g. \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ f' \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ g \ \text{andf} \ f \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ g \ \text{andf} \ f' \)

[eqf_controls] 
\( \vdash \forall M \ Oi \ Os \ P \ f \ f'. \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ f' \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ P \ \text{controls} \ f \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ P \ \text{controls} \ f' \)

[eqf_eq] 
\( \vdash (\text{Efn} \ Oi \ Os \ M \ (f_1 \ \text{eqf} \ f_2) = U (\text{b})) \ \iff \ \\
(\text{Efn} \ Oi \ Os \ M \ f_1 = \text{Efn} \ Oi \ Os \ M \ f_2) \)

[eqf_eqf1] 
\( \vdash \forall M \ Oi \ Os \ f \ f' \ g. \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ f' \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ g \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ f' \ \text{eqf} \ g \)

[eqf_eqf2] 
\( \vdash \forall M \ Oi \ Os \ f \ f' \ g. \\
(M, Oi, Os) \ \text{sat} \ f \ \text{eqf} \ f' \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ g \ \text{eqf} \ f \ \Rightarrow \\
(M, Oi, Os) \ \text{sat} \ g \ \text{eqf} \ f' \)
Theorems

ACLRULES THEORY

[eqf_impf1]
\[\vdash \forall M \ Oi \ Os \ f \ f' \ g.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } f \text{ impf } g \Rightarrow\]
\[(M,Oi,Os) \text{ sat } f' \text{ impf } g\]

[eqf_impf2]
\[\vdash \forall M \ Oi \ Os \ f \ f' \ g.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } g \text{ impf } f \Rightarrow\]
\[(M,Oi,Os) \text{ sat } g \text{ impf } f'\]

[eqf_notf]
\[\vdash \forall M \ Oi \ Os \ f \ f'.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } \text{ notf } f \Rightarrow\]
\[(M,Oi,Os) \text{ sat } \text{ notf } f'\]

[eqf_orf1]
\[\vdash \forall M \ Oi \ Os \ f \ f' \ g.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } f \text{ orf } g \Rightarrow\]
\[(M,Oi,Os) \text{ sat } f' \text{ orf } g\]

[eqf_orf2]
\[\vdash \forall M \ Oi \ Os \ f \ f' \ g.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } g \text{ orf } f \Rightarrow\]
\[(M,Oi,Os) \text{ sat } g \text{ orf } f'\]

[eqf_reps]
\[\vdash \forall M \ Oi \ Os \ P \ Q \ f \ f'.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } \text{ reps } P \ Q \ f \Rightarrow\]
\[(M,Oi,Os) \text{ sat } \text{ reps } P \ Q \ f'\]

[eqf_sat]
\[\vdash \forall M \ Oi \ Os \ f_1 \ f_2.\]
\[(M,Oi,Os) \text{ sat } f_1 \text{ eqf } f_2 \Rightarrow\]
\[
((M,Oi,Os) \text{ sat } f_1 \iff (M,Oi,Os) \text{ sat } f_2)\]

[eqf_says]
\[\vdash \forall M \ Oi \ Os \ P \ f \ f'.\]
\[(M,Oi,Os) \text{ sat } f \text{ eqf } f' \Rightarrow\]
\[(M,Oi,Os) \text{ sat } P \text{ says } f \Rightarrow\]
\[(M,Oi,Os) \text{ sat } P \text{ says } f'\]
\( [\text{eqi}_\text{Eq}] \)
\[ \vdash \forall M \, Oi \, Os \, l_1 \, l_2. \]
\[ (M, Oi, Os) \text{ sat } l_1 \text{ eqi } l_2 \iff (M, Oi, Os) \text{ sat } l_2 \text{ domi } l_1 \text{ andf } l_1 \text{ domi } l_2 \]

\( [\text{eqs}_\text{Eq}] \)
\[ \vdash \forall M \, Oi \, Os \, l_1 \, l_2. \]
\[ (M, Oi, Os) \text{ sat } l_1 \text{ eqs } l_2 \iff (M, Oi, Os) \text{ sat } l_2 \text{ doms } l_1 \text{ andf } l_1 \text{ doms } l_2 \]

\( [\text{Idemp}_\text{Speaks}_\text{For}] \)
\[ \vdash \forall M \, Oi \, Os \, P. \, (M, Oi, Os) \text{ sat } P \text{ speaks_for } P \]

\( [\text{Image}_\text{cmp}] \)
\[ \vdash \forall R_1 \, R_2 \, R_3 \, u. \, (R_1 \circ R_2) \, u \subseteq R_3 \iff R_2 \, u \subseteq \{ y \mid R_1 \, y \subseteq R_3 \} \]

\( [\text{Image}_\text{SUBSET}] \)
\[ \vdash \forall R_1 \, R_2. \, R_2 \text{ RSUBSET } R_1 \Rightarrow \forall w. \, R_2 \, w \subseteq R_1 \, w \]

\( [\text{Image}_\text{UNION}] \)
\[ \vdash \forall R_1 \, R_2 \, w. \, (R_1 \, \text{RUNION} \, R_2) \, w = R_1 \, w \cup R_2 \, w \]

\( [\text{INTER}_\text{EQ}_\text{UNIV}] \)
\[ \vdash (s \cap t = \mathcal{U}(\text{'}a\text{'))) \iff (s = \mathcal{U}(\text{'}a\text{'))} \land (t = \mathcal{U}(\text{'}a\text{'))) \]

\( [\text{Modus}_\text{Ponens}] \)
\[ \vdash \forall M \, Oi \, Os \, f_1 \, f_2. \]
\[ (M, Oi, Os) \text{ sat } f_1 \Rightarrow (M, Oi, Os) \text{ sat } f_1 \text{ impf } f_2 \Rightarrow (M, Oi, Os) \text{ sat } f_2 \]

\( [\text{Mono}_\text{speaks}_\text{for}] \)
\[ \vdash \forall M \, Oi \, Os \, P \, P' \, Q \, Q'. \]
\[ (M, Oi, Os) \text{ sat } P \text{ speaks_for } P' \Rightarrow (M, Oi, Os) \text{ sat } Q \text{ speaks_for } Q' \Rightarrow (M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ speaks_for } P' \text{ quoting } Q' \]

\( [\text{MP}_\text{ Says}] \)
\[ \vdash \forall M \, Oi \, Os \, P \, f_1 \, f_2. \]
\[ (M, Oi, Os) \text{ sat } \quad P \text{ says } (f_1 \text{ impf } f_2) \text{ impf } P \text{ says } f_1 \text{ impf } P \text{ says } f_2 \]

\( [\text{Quoting}] \)
\[ \vdash \forall M \, Oi \, Os \, P \, Q \, f. \]
\[ (M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ says } f \text{ eqf } P \text{ says } Q \text{ says } f \]
Theorems

ACLRULES THEORY

[Quoting_Eq]
\[ \vdash \forall M \, O_i \, O_s \, P \, Q \, f. \quad (M, O_i, O_s) \text{ sat } P \text{ quoting } Q \text{ says } f \iff (M, O_i, O_s) \text{ sat } P \text{ says } Q \text{ says } f \]

[reps_def_lemma]
\[ \vdash \forall M \, O_i \, O_s \, P \, Q \, f. \quad \text{Efn } O_i \, O_s \, M \text{ (reps } P \, Q \, f) = \text{Efn } O_i \, O_s \, M \text{ (P quoting } Q \text{ says } f \text{ impf } Q \text{ says } f) \]

[Reps_Eq]
\[ \vdash \forall M \, O_i \, O_s \, P \, Q \, f. \quad (M, O_i, O_s) \text{ sat reps } P \, Q \, f \iff (M, O_i, O_s) \text{ sat } P \text{ quoting } Q \text{ says } f \text{ impf } Q \text{ says } f \]

[sat_allworld]
\[ \vdash \forall f. \quad (M, O_i, O_s) \text{ sat } f \iff \forall w. \ w \in \text{Efn } O_i \, O_s \, M \, f \]

[sat_andf_eq_and_sat]
\[ \vdash (M, O_i, O_s) \text{ sat } f_1 \text{ andf } f_2 \iff (M, O_i, O_s) \text{ sat } f_1 \land (M, O_i, O_s) \text{ sat } f_2 \]

[sat_TT]
\[ \vdash (M, O_i, O_s) \text{ sat TT} \]

[Says]
\[ \vdash \forall M \, O_i \, O_s \, P \, f. \quad (M, O_i, O_s) \text{ sat } f \Rightarrow (M, O_i, O_s) \text{ sat } P \text{ says } f \]

[says_and_lemma]
\[ \vdash \forall M \, O_i \, O_s \, P \, Q \, f. \quad (M, O_i, O_s) \text{ sat } P \text{ says } f \text{ andf } Q \text{ says } f \text{ impf } P \text{ meet } Q \text{ says } f \]

[Speaks_For]
\[ \vdash \forall M \, O_i \, O_s \, P \, Q \, f. \quad (M, O_i, O_s) \text{ sat } P \text{ speaks_for } Q \text{ impf } P \text{ says } f \text{ impf } Q \text{ says } f \]

[Speaks_for_SUBSET]
\[ \vdash \forall R_3 \, R_2 \, R_1. \quad R_2 \text{ SINSET } R_1 \Rightarrow \forall w. \ \{ w \mid R_1 \ w \subseteq R_3 \} \subseteq \{ w \mid R_2 \ w \subseteq R_3 \} \]

[SUBSET_Image_SUBSET]
\[ \vdash \forall R_1 \, R_2 \, R_3. \quad (\forall w_1. \ R_2 \ w_1 \subseteq R_1 \ w_1) \Rightarrow \forall w. \ \{ w \mid R_1 \ w \subseteq R_3 \} \subseteq \{ w \mid R_2 \ w \subseteq R_3 \} \]
[**Trans_Speaks_For**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; P \; Q \; R.
\]
\[
(M, \text{Oi}, \text{Os}) \text{ sat } P \text{ speaks_for } Q \Rightarrow
\]
\[
(M, \text{Oi}, \text{Os}) \text{ sat } Q \text{ speaks_for } R \Rightarrow
\]
\[
(M, \text{Oi}, \text{Os}) \text{ sat } P \text{ speaks_for } R
\]

[**UNIV_DIFF_SUBSET**]
\[
\vdash \forall R_1 \; R_2. \; R_1 \subseteq R_2 \Rightarrow (\mathcal{U}(\text{a}) \text{ DIFF } R_1 \cup R_2 = \mathcal{U}(\text{a}))
\]

[**world_and**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; f_1 \; f_2 \; w.
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; (f_1 \; \text{andf} \; f_2) \iff
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_1 \land w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_2
\]

[**world_eq**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; f_1 \; f_2 \; w.
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; (f_1 \; \text{eqf} \; f_2) \iff
\]
\[
(w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_1 \iff w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_2)
\]

[**world_eqn**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; n_1 \; n_2 \; w. \; w \in \text{Efn} \; \text{Oi} \; \text{Os} \; m \; (n_1 \; \text{eqn} \; n_2) \iff (n_1 = n_2)
\]

[**world_F**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; w. \; w \notin \text{Efn} \; \text{Oi} \; \text{Os} \; M \; \text{FF}
\]

[**world_imp**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; f_1 \; f_2 \; w.
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; (f_1 \; \text{impf} \; f_2) \iff
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_1 \Rightarrow w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_2
\]

[**world_lt**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; n_1 \; n_2 \; w. \; w \in \text{Efn} \; \text{Oi} \; \text{Os} \; m \; (n_1 \; \text{lt} \; n_2) \iff n_1 < n_2
\]

[**world_lte**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; n_1 \; n_2 \; w. \; w \in \text{Efn} \; \text{Oi} \; \text{Os} \; m \; (n_1 \; \text{lte} \; n_2) \iff n_1 \leq n_2
\]

[**world_not**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; f \; w. \; w \in \text{Efn} \; \text{Oi} \; \text{Os} \; m \; (\text{notf} \; f) \iff w \notin \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f
\]

[**world_or**]
\[
\vdash \forall M \; f_1 \; f_2 \; w.
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; (f_1 \; \text{orf} \; f_2) \iff
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_1 \lor w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f_2
\]

[**world_says**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; P \; f \; w.
\]
\[
w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; (P \; \text{says} \; f) \iff
\]
\[
\forall v. \; v \in \text{Jext} \; (\text{jKS} \; M) \; P \; w \Rightarrow v \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; f
\]

[**world_T**]
\[
\vdash \forall M \; \text{Oi} \; \text{Os} \; w. \; w \in \text{Efn} \; \text{Oi} \; \text{Os} \; M \; \text{TT}
\]
4 aclDrules Theory

Built: 25 February 2018
Parent Theories: aclrules

4.1 Theorems

[Conjunction]
\[\vdash \forall M \ Oi \ Os \ f_1 \ f_2.\]
\[(M,Oi,Os) \ sat f_1 \Rightarrow\]
\[(M,Oi,Os) \ sat f_2 \Rightarrow\]
\[(M,Oi,Os) \ sat f_1 \ and f_2\]

[Controls]
\[\vdash \forall M \ Oi \ Os \ P \ f.\]
\[(M,Oi,Os) \ sat P \ says f \Rightarrow\]
\[(M,Oi,Os) \ sat P \ controls f \Rightarrow\]
\[(M,Oi,Os) \ sat f\]

[Derived_Controls]
\[\vdash \forall M \ Oi \ Os \ P \ Q \ f.\]
\[(M,Oi,Os) \ sat P \ speaks_for Q \Rightarrow\]
\[(M,Oi,Os) \ sat Q \ controls f \Rightarrow\]
\[(M,Oi,Os) \ sat P \ controls f\]

[Derived_Speaks_For]
\[\vdash \forall M \ Oi \ Os \ P \ Q \ f.\]
\[(M,Oi,Os) \ sat P \ speaks_for Q \Rightarrow\]
\[(M,Oi,Os) \ sat P \ says f \Rightarrow\]
\[(M,Oi,Os) \ sat Q \ says f\]

[Disjunction1]
\[\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M,Oi,Os) \ sat f_1 \Rightarrow (M,Oi,Os) \ sat f_1 \ or f_2\]

[Disjunction2]
\[\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M,Oi,Os) \ sat f_2 \Rightarrow (M,Oi,Os) \ sat f_1 \ or f_2\]

[Disjunctive_Syllogism]
\[\vdash \forall M \ Oi \ Os \ f_1 \ f_2.\]
\[(M,Oi,Os) \ sat f_1 \ or f_2 \Rightarrow\]
\[(M,Oi,Os) \ sat not f_1 \Rightarrow\]
\[(M,Oi,Os) \ sat f_2\]

[Double_Negation]
\[\vdash \forall M \ Oi \ Os \ f. \ (M,Oi,Os) \ sat not f \ (not f) \Rightarrow (M,Oi,Os) \ sat f\]
[/eqn_eqn]
\[ (M, O_i, O_s) \text{ sat } c_1 \text{ eqn } n_1 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_2 \text{ eqn } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } n_1 \text{ eqn } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_1 \text{ eqn } c_2 \]

[/eqn_lt]
\[ (M, O_i, O_s) \text{ sat } c_1 \text{ eqn } n_1 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_2 \text{ eqn } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } n_1 \text{ lt } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_1 \text{ lt } c_2 \]

[/eqn_lte]
\[ (M, O_i, O_s) \text{ sat } c_1 \text{ eqn } n_1 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_2 \text{ eqn } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } n_1 \text{ lte } n_2 \Rightarrow \\
(M, O_i, O_s) \text{ sat } c_1 \text{ lte } c_2 \]

[Hypothetical_Syllogism]
\[ \forall M O_i O_s f_1 f_2 f_3. \\
\quad (M, O_i, O_s) \text{ sat } f_1 \text{ impf } f_2 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } f_2 \text{ impf } f_3 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } f_1 \text{ impf } f_3 \]

/il_domi]
\[ \forall M O_i O_s P Q l_1 l_2. \\
\quad (M, O_i, O_s) \text{ sat } \text{il } P \text{ eqi } l_1 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } \text{il } Q \text{ eqi } l_2 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } l_2 \text{ domi } l_1 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } \text{il } Q \text{ domi } \text{il } P \]

[INTER_EQ_UNIV]
\[ \forall s_1 s_2. \ (s_1 \cap s_2 = U(:'a)) \iff (s_1 = U(:'a)) \land (s_2 = U(:'a)) \]

[Modus_Tollens]
\[ \forall M O_i O_s f_1 f_2. \\
\quad (M, O_i, O_s) \text{ sat } f_1 \text{ impf } f_2 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } \text{notf } f_2 \Rightarrow \\
\quad (M, O_i, O_s) \text{ sat } \text{notf } f_1 \]

[Rep_Controls_Eq]
\[ \forall M O_i O_s A B f. \\
\quad (M, O_i, O_s) \text{ sat } \text{reps } A B f \iff \\
\quad (M, O_i, O_s) \text{ sat } A \text{ controls } B \text{ says } f \]
\[\text{Rep\_Says}\]
\[
\vdash \forall M \ Oi \ Os \ P \ Q \ f.
(M,Oi,Os) \ sats \ rep \ P \ Q \ f \Rightarrow
(M,Oi,Os) \ sats \ P \ quoting \ Q \ says \ f \Rightarrow
(M,Oi,Os) \ sats \ Q \ says \ f
\]

\[\text{Reps}\]
\[
\vdash \forall M \ Oi \ Os \ P \ Q \ f.
(M,Oi,Os) \ sats \ rep \ P \ Q \ f \Rightarrow
(M,Oi,Os) \ sats \ P \ quoting \ Q \ says \ f \Rightarrow
(M,Oi,Os) \ sats \ Q \ controls \ f \Rightarrow
(M,Oi,Os) \ sats \ f
\]

\[\text{Says\_Simplification1}\]
\[
\vdash \forall M \ Oi \ Os \ P \ f_1 \ f_2.
(M,Oi,Os) \ sats \ P \ says \ (f_1 \ andf \ f_2) \Rightarrow (M,Oi,Os) \ sats \ P \ says \ f_1
\]

\[\text{Says\_Simplification2}\]
\[
\vdash \forall M \ Oi \ Os \ P \ f_1 \ f_2.
(M,Oi,Os) \ sats \ P \ says \ (f_1 \ andf \ f_2) \Rightarrow (M,Oi,Os) \ sats \ P \ says \ f_2
\]

\[\text{Simplification1}\]
\[
\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M,Oi,Os) \ sats \ f_1 \ andf \ f_2 \Rightarrow (M,Oi,Os) \ sats \ f_1
\]

\[\text{Simplification2}\]
\[
\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M,Oi,Os) \ sats \ f_1 \ andf \ f_2 \Rightarrow (M,Oi,Os) \ sats \ f_2
\]

\[\text{sl\_doms}\]
\[
\vdash \forall M \ Oi \ Os \ P \ Q \ l_1 \ l_2.
(M,Oi,Os) \ sats \ sl \ P \ eqs \ l_1 \Rightarrow
(M,Oi,Os) \ sats \ sl \ Q \ eqs \ l_2 \Rightarrow
(M,Oi,Os) \ sats \ l_2 \ doms \ l_1 \Rightarrow
(M,Oi,Os) \ sats \ sl \ Q \ doms \ sl \ P
\]
Index

aclDrules Theory, 17
 Theorems, 17
 Conjunction, 17
 Controls, 17
 Derived Controls, 17
 Derived Speaks For, 17
 Disjunction1, 17
 Disjunction2, 17
 Disjunctive Syllogism, 17
 Double Negation, 17
 eqn_eqn, 18
 eqn_lt, 18
 eqn_lte, 18
 Hypothetical Syllogism, 18
 iI_domi, 18
 INTER_EQ_UNIV, 18
 Modus Tollens, 18
 Rep Controls Eq, 18
 Rep Says, 19
 Reps, 19
 Says Simplification1, 19
 Says Simplification2, 19
 Simplification1, 19
 Simplification2, 19
 sl_doms, 19

aclrules Theory, 10
 Definitions, 11
 sat def, 11
 Theorems, 11
 And Says, 11
 And Says Eq, 11
 and says lemma, 11
 Controls Eq, 11
 DIFF_UNIV_SUBSET, 11
 domi antisymmetric, 11
 domi reflexive, 11
 domi transitive, 11
 doms antisymmetric, 11
 doms reflexive, 12
 doms transitive, 12
 eqf and impf, 12
 eqf andf1, 12
 eqf andf2, 12
 eqf controls, 12
 eqf eq, 12
 eqf eqf1, 12

aclfoundation Theory, 3
 Datatypes, 3
 Definitions, 3
 imapKS def, 4
 inptKS def, 4
 jKS def, 4
 O1 def, 4
 one weakorder def, 4
 po TY DEF, 4
 po tybij, 4
 prod PO def, 4
 smapKS def, 4
 Subset PO def, 4
 Theorems, 4
 abs po11, 4

absPO fn onto, 4
 antisym prod antisym, 5
 EQ WeakOrder, 5
 KS bij, 5
 one weakorder WO, 5
 onto po, 5
 po bij, 5
 PO repPO, 5
 refl prod refl, 5
 repPO iPO partial order, 5
 repPO O1, 5
 repPO prod PO, 5
 repPO Subset PO, 5
 RPROD THM, 5
 SUBSET WO, 6
 trans prod trans, 6
 WeakOrder Exists, 6
 WO prod WO, 6
 WO repPO, 6
eqf_eqf2, 12
eqf_\text{impf1}, 13
eqf_\text{impf2}, 13
eqf_\text{notf}, 13
eqf_\text{orf1}, 13
eqf_\text{orf2}, 13
eqf_\text{reps}, 13
\equivf_\text{sat}, 13
\equivf_\text{says}, 13
\equivf_\text{Eq}, 14
\equivf_\text{eqs}, 14
\text{Idemp}_\text{Speaks}_\text{For}, 14
\text{Image}_\text{cmp}, 14
\text{Image}_\text{SUBSET}, 14
\text{Image}_\text{UNION}, 14
\text{INTER}_\text{EQ}_\text{UNIV}, 14
\text{Modus}_\text{Ponens}, 14
\text{Mono}_\text{speaks}_\text{for}, 14
\text{MP}_\text{Says}, 14
\text{Quoting}, 14
\text{Quoting}_\text{Eq}, 15
\text{reps}_\text{def}_\text{lemma}, 15
\text{Reps}_\text{Eq}, 15
\text{sat}_\text{allworld}, 15
\text{sat}_\text{andf}_\text{eq}_\text{and}_\text{sat}, 15
\text{sat}_\text{TT}, 15
\text{Says}, 15
\text{says}_\text{and}_\text{lemma}, 15
\text{Speaks}_\text{For}, 15
\text{speaks}_\text{for}_\text{SUBSET}, 15
\text{SUBSET}_\text{Image}_\text{SUBSET}, 15
\text{Trans}_\text{Speaks}_\text{For}, 16
\text{UNIV}_\text{DIFF}_\text{SUBSET}, 16
\text{world}_\text{and}, 16
\text{world}_\text{eq}, 16
\text{world}_\text{eqn}, 16
\text{world}_\text{F}, 16
\text{world}_\text{imp}, 16
\text{world}_\text{lt}, 16
\text{world}_\text{lte}, 16
\text{world}_\text{not}, 16
\text{world}_\text{or}, 16
\text{world}_\text{says}, 16
\text{world}_\text{T}, 16
\text{aclsemantics}_\text{Theory}, 6
\text{Definitions}, 6
\text{Efn}_\text{def}, 6
\text{Jext}_\text{def}, 7
\text{Lifn}_\text{def}, 7
\text{Lsfn}_\text{def}, 8
\text{Theorems}, 8
\text{andf}_\text{def}, 8
\text{controls}_\text{def}, 8
\text{controls}_\text{says}, 8
\text{domi}_\text{def}, 8
\text{doms}_\text{def}, 8
\text{eqf}_\text{def}, 8
\text{eqf}_\text{impf}, 8
\text{eqf}_\text{def}, 9
\text{eqf}_\text{domi}, 9
\text{eqn}_\text{def}, 9
\text{eqs}_\text{def}, 9
\text{eqs}_\text{doms}, 9
\text{FF}_\text{def}, 9
\text{impf}_\text{def}, 9
\text{lt}_\text{def}, 9
\text{lte}_\text{def}, 9
\text{meet}_\text{def}, 10
\text{name}_\text{def}, 10
\text{notf}_\text{def}, 10
\text{orf}_\text{def}, 10
\text{prop}_\text{def}, 10
\text{quoting}_\text{def}, 10
\text{reps}_\text{def}, 10
\text{says}_\text{def}, 10
\text{speaks}_\text{for}_\text{def}, 10
\text{TT}_\text{def}, 10
Appendix B

Parametrizable Secure State Machine & Patrol Base Operations: Pretty-Printed Theories
## Contents

1. **OMNIType Theory** | 3  
   1.1 Datatypes | 3  
   1.2 Theorems | 3  

2. **ssm11 Theory** | 4  
   2.1 Datatypes | 4  
   2.2 Definitions | 4  
   2.3 Theorems | 5  

3. **ssm Theory** | 11  
   3.1 Datatypes | 11  
   3.2 Definitions | 12  
   3.3 Theorems | 13  

4. **satList Theory** | 21  
   4.1 Definitions | 21  
   4.2 Theorems | 21  

5. **PBTtypeIntegrated Theory** | 21  
   5.1 Datatypes | 21  
   5.2 Theorems | 22  

6. **PBIntegratedDef Theory** | 23  
   6.1 Definitions | 23  
   6.2 Theorems | 24  

7. **ssmPBIntegrated Theory** | 28  
   7.1 Theorems | 28  

8. **ssmConductORP Theory** | 35  
   8.1 Theorems | 35  

9. **ConductORPType Theory** | 44  
   9.1 Datatypes | 44  
   9.2 Theorems | 45  

10. **ConductORPDdef Theory** | 46  
    10.1 Definitions | 46  
    10.2 Theorems | 47  

11. **ssmConductPB Theory** | 51  
    11.1 Definitions | 51  
    11.2 Theorems | 52
1 OMNIType Theory

Built: 10 June 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

\( \text{command} = \text{ESCc} \ \text{escCommand} \ | \ \text{SLc} \ 'slCommand \)

\( \text{escCommand} = \text{returnToBase} \ | \ \text{changeMission} \ | \ \text{resupply} \ | \ \text{reactToContact} \)

\( \text{escOutput} = \text{ReturnToBase} \ | \ \text{ChangeMission} \ | \ \text{Resupply} \ | \ \text{ReactToContact} \)

\( \text{escState} = \text{RTB} \ | \ \text{CM} \ | \ \text{RESUPPLY} \ | \ \text{RTC} \)

\( \text{output} = \text{ESCo} \ \text{escOutput} \ | \ \text{SLo} \ 'slOutput \)

\( \text{principal} = \text{SR} \ 'stateRole \)

\( \text{state} = \text{ESCs} \ \text{escState} \ | \ \text{SLs} \ 'slState \)

1.2 Theorems

[\text{command\_distinct\_clauses}]
\[ \forall a \ a'. \ \text{ESCc} \ a \neq \text{SLc} \ a' \]

[\text{command\_one\_one}]
\[ \vdash (\forall a \ a'. \ (\text{ESCc} \ a = \text{ESCc} \ a') \iff (a = a')) \wedge \\
\forall a \ a'. \ (\text{SLc} \ a = \text{SLc} \ a') \iff (a = a') \]

[\text{escCommand\_distinct\_clauses}]
\[ \vdash \text{returnToBase} \neq \text{changeMission} \wedge \text{returnToBase} \neq \text{resupply} \wedge \\
\text{returnToBase} \neq \text{reactToContact} \wedge \text{changeMission} \neq \text{resupply} \wedge \\
\text{changeMission} \neq \text{reactToContact} \wedge \text{resupply} \neq \text{reactToContact} \]

[\text{escOutput\_distinct\_clauses}]
\[ \vdash \text{ReturnToBase} \neq \text{ChangeMission} \wedge \text{ReturnToBase} \neq \text{Resupply} \wedge \\
\text{ReturnToBase} \neq \text{ReactToContact} \wedge \text{ChangeMission} \neq \text{Resupply} \wedge \\
\text{ChangeMission} \neq \text{ReactToContact} \wedge \text{Resupply} \neq \text{ReactToContact} \]

[\text{escState\_distinct\_clauses}]
\[ \vdash \text{RTB} \neq \text{CM} \wedge \text{RTB} \neq \text{RESUPPLY} \wedge \text{RTB} \neq \text{RTC} \wedge \text{CM} \neq \text{RESUPPLY} \wedge \\
\text{CM} \neq \text{RTC} \wedge \text{RESUPPLY} \neq \text{RTC} \]
[output_distinct_clauses]
\[\forall a\ a\ E \neq S\ a\]

[output_one_one]
\[\forall a\ a\cdot (E = E) \iff (a = a\prime) \land
   \forall a\ a\cdot (S = S) \iff (a = a\prime)\]

[principal_one_one]
\[\forall a\ a\cdot (P = P) \iff (a = a\prime)\]

[state_distinct_clauses]
\[\forall a\ a\ E \neq S\ a\]

[state_one_one]
\[\forall a\ a\cdot (E = E) \iff (a = a\prime) \land
   \forall a\ a\cdot (S = S) \iff (a = a\prime)\]

2 ssm11 Theory

Built: 10 June 2018

Parent Theories: satList

2.1 Datatypes

configuration =
  CFG (('command order, 'principal, 'd, 'e) Form -> bool)
   ('state -> ('command order, 'principal, 'd, 'e) Form)
   ('command order, 'principal, 'd, 'e) Form list)
   ('command order, 'principal, 'd, 'e) Form list) 'state
   ('output list)

order = SOME 'command | NONE

trType = discard 'command | trap 'command | exec 'command

2.2 Definitions

[TR_def]
\[=\]
\[\forall a\ a\cdot (a = a\prime) \land
   \forall a\ a\cdot (a = a\prime)\]

exists authenticationTest P NS M Oi Os Out s
   securityContext stateInterp cmd ins outs.
   (a0 = (M, Oi, Os)) \land (a1 = exec cmd) \land
   (a2 =


|
CFG authenticationTest stateInterp
securityContext \((P \text{ says prop (SOME cmd)::ins}) s\)
outs) ∧
\((a_3 =
CFG authenticationTest stateInterp
securityContext ins (NS s (exec cmd))
(Out s (exec cmd)::outs)) ∧
authenticationTest \((P \text{ says prop (SOME cmd)})\) ∧
CFGInterpret \((M, O_i, O_s)\)
(CFG authenticationTest stateInterp
securityContext \((P \text{ says prop (SOME cmd)::ins}) s\)
outs)) \lor
(∃authenticationTest \(P\) NS M Oi Os Out s
securityContext stateInterp cmd ins outs.
\((a_0 = (M, O_i, O_s)) \land (a_1 = \text{trap cmd}) \land
\((a_2 =
CFG authenticationTest stateInterp
securityContext \((P \text{ says prop (SOME cmd)::ins}) s\)
outs)) \land
\((a_3 =
CFG authenticationTest stateInterp
securityContext ins (NS s (trap cmd))
(Out s (trap cmd)::outs)) \land
authenticationTest \((P \text{ says prop (SOME cmd)})\) \land
CFGInterpret \((M, O_i, O_s)\)
(CFG authenticationTest stateInterp
securityContext \((P \text{ says prop (SOME cmd)::ins}) s\)
outs)) \lor
(∃authenticationTest NS M Oi Os Out s securityContext
stateInterp cmd x ins outs.
\((a_0 = (M, O_i, O_s)) \land (a_1 = \text{discard cmd}) \land
\((a_2 =
CFG authenticationTest stateInterp
securityContext \((x::ins) s\) outs)) \land
\((a_3 =
CFG authenticationTest stateInterp
securityContext ins (NS s (discard cmd))
(Out s (discard cmd)::outs)) \land
\neg authenticationTest \(x\) \Rightarrow
\text{TR}' a_0 a_1 a_2 a_3 \Rightarrow
\text{TR}' a_0 a_1 a_2 a_3)

\section*{2.3 Theorems}

\textbf{CFGInterpret_def}

\(\vdash \text{CFGInterpret} \ (M, O_i, O_s)
(CFG authenticationTest stateInterp securityContext
\text{input::ins} state outputStream) \iff\)
\((M, Oi, Os) \text{ satList securityContext } \land (M, Oi, Os) \text{ sat input } \land (M, Oi, Os) \text{ sat stateInterp state}\)

**CFGInterpret_ind**

\(\vdash \forall P.\)
\(\forall M \ Oi \ Os \ \text{authenticationTest stateInterp securityContext}
\input ins state outputStream.
\(P (M, Oi, Os)
\(\text{(CFG authenticationTest stateInterp securityContext}
\(\text{(input::ins) state outputStream})) \land
\(\forall v_1 v_2 v_3. \ P (v_1, v_2) v_3
\)

**configuration_one_one**

\(\vdash \forall a_0 a_1 a_2 a_3 a_4 a_5 a_6 a'_0 a'_1 a'_2 a'_3 a'_4 a'_5.
\(\text{(CFG } a_0 a_1 a_2 a_3 a_4 a_5 a_6 = \text{CFG } a'_0 a'_1 a'_2 a'_3 a'_4 a'_5) \iff
\(\text{(a}_0 = a'_0) \land (a_1 = a'_1) \land (a_2 = a'_2) \land (a_3 = a'_3) \land
\(\text{(a}_4 = a'_4) \land (a_5 = a'_5)
\)

**order_distinct_clauses**

\(\vdash \forall a. \ \text{SOME } a \neq \text{NONE}
\)

**order_one_one**

\(\vdash \forall a a'. \ (\text{SOME } a = \text{SOME } a') \iff (a = a')
\)

**TR_cases**

\(\vdash \forall a_0 a_1 a_2 a_3.
\(\text{TR } a_0 a_1 a_2 a_3 \iff
\(\exists \ \text{authenticationTest } P \ NS M Oi Os Out s \text{ securityContext}
\text{stateInterp cmd ins outs}.
\(\text{(a}_0 = (M, Oi, Os)) \land (a_1 = \text{exec cmd}) \land
\(\text{(a}_2 = \text{CFG authenticationTest stateInterp securityContext}
\text{(P says prop (SOME cmd)::ins) s outs}) \land
\(\text{(a}_3 = \text{CFG authenticationTest stateInterp securityContext ins}
\text{(NS s (exec cmd)) (Out s (exec cmd)::outs)) \land}
\text{authenticationTest (P says prop (SOME cmd))})
\text{\land}
\text{CFGInterpret (M, Oi, Os)}
\text{(CFG authenticationTest stateInterp securityContext}
\text{(P says prop (SOME cmd)::ins) s outs}) \land
\(\exists \ \text{authenticationTest } P \ NS M Oi Os Out s \text{ securityContext}
\text{stateInterp cmd ins outs}.
\(\text{(a}_0 = (M, Oi, Os)) \land (a_1 = \text{trap cmd}) \land
\(\text{(a}_2 = \text{CFG authenticationTest stateInterp securityContext}
\text{(P says prop (SOME cmd)::ins) s outs}) \land
\)
\( a_3 = \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( \text{(NS s (trap cmd)) (Out s (trap cmd)::outs)) \land \) authenticationTest \( (P \text{ says prop (SOME cmd)}) \land \)
\( \text{CFGInterpret (M, Oi, Os)} \)
\( \text{(CFG authenticationTest stateInterp securityContext ins} \)
\( \text{(P says prop (SOME cmd)::ins) s outs)) \land \) authenticationTest \( NS M Oi Os Out s \) securityContext
\( \text{stateInterp cmd x ins outs.} \)
\( a_0 = (M, Oi, Os)) \land (a_1 = \text{discard cmd}) \land \)
\( a_2 = \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (x::ins) s outs) \land \)
\( a_3 = \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (NS s (\text{discard cmd})) (Out s (\text{discard cmd)::outs})) \land \) 
\( \neg \text{authenticationTest x} \)

\[ \text{[TR_discard_cmd_rule]} \]
\( \vdash \text{TR (M, Oi, Os) (\text{discard cmd})} \)
\( \text{(CFG authenticationTest stateInterp securityContext ins} \)
\( (x::ins) s outs) \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (NS s (\text{exec cmd})) (Out s (\text{exec cmd)::outs})) \iff \neg \text{authenticationTest x} \)

\[ \text{[TR_EQ_rules_thm]} \]
\( \vdash \text{TR (M, Oi, Os) (exec cmd)} \)
\( \text{(CFG authenticationTest stateInterp securityContext ins} \)
\( (P \text{ says prop (SOME cmd)::ins) s outs}) \) 
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (NS s (\text{exec cmd})) (Out s (\text{exec cmd)::outs})) \iff \neg \text{authenticationTest x} \)
\( \text{CFGInterpret (M, Oi, Os)} \)
\( \text{(CFG authenticationTest stateInterp securityContext ins} \)
\( (P \text{ says prop (SOME cmd)::ins) s outs)} \land \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (NS s (\text{trap cmd})) (Out s (\text{trap cmd)::outs})) \iff \neg \text{authenticationTest x} \)
\( \text{CFGInterpret (M, Oi, Os)} \)
\( \text{(CFG authenticationTest stateInterp securityContext ins} \)
\( (P \text{ says prop (SOME cmd)::ins) s outs}) \land \)
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
\( (x::ins) s outs) \) 
\( \text{CFG authenticationTest stateInterp securityContext ins} \)
(NS s (discard cmd)) (Out s (discard cmd)::outs)) \iff\neg\text{authenticationTest } x
\end{equation*}

\[\text{TR\_exec\_cmd\_rule}\]
\[\vdash \forall \text{authenticationTest securityContext stateInterp } P \text{ cmd ins s outs}.
(\forall M Oi Os.
CFGInterpret (M,Oi,Os)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs) \Rightarrow
(M,Oi,Os) sat prop (SOME cmd)) \Rightarrow
\forall NS Out M Oi Os.
TR (M,Oi,Os) (exec cmd)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs)
(CFG authenticationTest stateInterp securityContext ins
(NS s (exec cmd)) (Out s (exec cmd)::outs)) \iff authenticationTest (P says prop (SOME cmd)) \land
CFGInterpret (M,Oi,Os)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs) \land
(M,Oi,Os) sat prop (SOME cmd)
\]

\[\text{TR\_ind}\]
\[\vdash \forall TR'.
(\forall \text{authenticationTest } P \text{ NS } M \text{ Oi } Os \text{ Out } s \text{ securityContext stateInterp } \text{ cmd ins outs}.
authenticationTest (P says prop (SOME cmd)) \land
CFGInterpret (M,Oi,Os)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs) \Rightarrow
TR' (M,Oi,Os) (exec cmd)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs)
(CFG authenticationTest stateInterp securityContext ins
(NS s (exec cmd)) (Out s (exec cmd)::outs))) \land
(\forall \text{authenticationTest } P \text{ NS } M \text{ Oi } Os \text{ Out } s \text{ securityContext stateInterp } \text{ cmd ins outs}.
authenticationTest (P says prop (SOME cmd)) \land
CFGInterpret (M,Oi,Os)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs) \Rightarrow
TR' (M,Oi,Os) (trap cmd)
(CFG authenticationTest stateInterp securityContext
(P says prop (SOME cmd)::ins) s outs)
(CFG authenticationTest stateInterp securityContext ins
(NS s (trap cmd)) (Out s (trap cmd)::outs))) \land
(\forall \text{authenticationTest } NS \text{ M } Oi \text{ Os } Out s \text{ securityContext stateInterp } \text{ cmd } x \text{ ins outs}.
\]
\[\neg \text{authenticationTest } x \Rightarrow\]
\[TR' (M, Oi, Os) \text{ (discard cmd)}\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(x::ins) s outs)\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(ins (NS s (discard cmd)))\]
\[(Out s (discard cmd)::outs)) \Rightarrow\]
\[\forall a_0 \ a_1 \ a_2 \ a_3. \ TR \ a_0 \ a_1 \ a_2 \ a_3 \Rightarrow TR' \ a_0 \ a_1 \ a_2 \ a_3\]

\[\textbf{[TR\_rules]}\]
\[\vdash (\forall \text{authenticationTest } P \ NS \ M \ Oi \ Os \ Out s \ securityContext \]
\[\text{stateInterp cmd ins outs.}\]
\[\text{authenticationTest } (P \ says \ prop \ (\text{SOME cmd})) \land \]
\[\text{CFGInterpret } (M, Oi, Os)\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(P \ says \ prop \ (\text{SOME cmd})::ins) s outs) \Rightarrow\]
\[\text{TR } (M, Oi, Os) \text{ (exec cmd)}\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(P \ says \ prop \ (\text{SOME cmd})::ins) s outs)\]
\[(\text{CFG authenticationTest stateInterp securityContext ins}\]
\[(NS s (exec cmd)) \ (Out s (exec cmd)::outs)) \land \]
\[(\forall \text{authenticationTest } P \ NS \ M \ Oi \ Os \ Out s \ securityContext \]
\[\text{stateInterp cmd ins outs.}\]
\[\text{authenticationTest } (P \ says \ prop \ (\text{SOME cmd})) \land \]
\[\text{CFGInterpret } (M, Oi, Os)\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(P \ says \ prop \ (\text{SOME cmd})::ins) s outs) \Rightarrow\]
\[\text{TR } (M, Oi, Os) \text{ (trap cmd)}\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(P \ says \ prop \ (\text{SOME cmd})::ins) s outs)\]
\[(\text{CFG authenticationTest stateInterp securityContext ins}\]
\[(NS s (\text{trap cmd})) \ (Out s (\text{trap cmd})::outs)) \land \]
\[(\forall \text{authenticationTest } NS \ M \ Oi \ Os \ Out s \ securityContext \]
\[\text{stateInterp cmd x ins outs.}\]
\[\neg \text{authenticationTest } x \Rightarrow\]
\[TR (M, Oi, Os) \text{ (discard cmd)}\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(x::ins) s outs)\]
\[(\text{CFG authenticationTest stateInterp securityContext ins}\]
\[(NS s (\text{discard cmd})) \ (Out s (\text{discard cmd})::outs))\]

\[\textbf{[TR\_strongind]}\]
\[\vdash \forall TR'.\]
\[(\forall \text{authenticationTest } P \ NS \ M \ Oi \ Os \ Out s \ securityContext \]
\[\text{stateInterp cmd ins outs.}\]
\[\text{authenticationTest } (P \ says \ prop \ (\text{SOME cmd})) \land \]
\[\text{CFGInterpret } (M, Oi, Os)\]
\[(\text{CFG authenticationTest stateInterp securityContext}\]
\[(P \ says \ prop \ (\text{SOME cmd})::ins) s outs) \Rightarrow\]
\[ TR' (M, Oi, Os) \ (\text{exec cmd}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (P \text{ says prop (SOME cmd)}::\text{ins} s \text{ outs}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ \text{ins} (NS s (\text{exec cmd})) (\text{Out s (exec cmd)}::\text{outs})) \wedge \]
\[ (\forall \text{authenticationTest} P \ NS M Oi Os \text{ Out s securityContext} \]
\[ \text{stateInterp cmd ins outs,} \]
\[ \text{authenticationTest} (P \text{ says prop (SOME cmd)}) \wedge \]
\[ \text{CFGInterpret} (M, Oi, Os) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (P \text{ says prop (SOME cmd)}::\text{ins} s \text{ outs}) \Rightarrow \]
\[ TR' (M, Oi, Os) \ (\text{trap cmd}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ \text{ins} (NS s (\text{trap cmd})) (\text{Out s (trap cmd)}::\text{outs})) \wedge \]
\[ (\forall \text{authenticationTest} NS M Oi Os \text{ Out s securityContext} \]
\[ \text{stateInterp cmd x ins outs,} \]
\[ \neg \text{authenticationTest} x \Rightarrow \]
\[ TR' (M, Oi, Os) \ (\text{discard cmd}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (x::\text{ins}) s \text{ outs}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ \text{ins} (NS s (\text{discard cmd})) \]
\[ (\text{Out s (discard cmd)}::\text{outs})) \Rightarrow \]
\[ \forall a_0 a_1 a_2 a_3 \text{. TR a_0 a_1 a_2 a_3 } \Rightarrow TR' a_0 a_1 a_2 a_3 \]

\[ \text{[TR_trap_cmd_rule]} \]
\[ \vdash \forall \text{authenticationTest stateInterp securityContext} P \text{ cmd ins s outs.} \]
\[ (\forall M Oi Os, \]
\[ \text{CFGInterpret} (M, Oi, Os) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (P \text{ says prop (SOME cmd)}::\text{ins} s \text{ outs}) \Rightarrow \]
\[ (M, Oi, Os) \text{ sat prop NONE } \Rightarrow \]
\[ \forall NS Out M Oi Os, \]
\[ \text{TR} (M, Oi, Os) \ (\text{trap cmd}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (P \text{ says prop (SOME cmd)}::\text{ins} s \text{ outs}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ \text{ins} (NS s (\text{trap cmd})) (\text{Out s (trap cmd)}::\text{outs})) \iff \]
\[ \text{authenticationTest} (P \text{ says prop (SOME cmd)}) \wedge \]
\[ \text{CFGInterpret} (M, Oi, Os) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]
\[ (P \text{ says prop (SOME cmd)}::\text{ins} s \text{ outs}) \wedge \]
\[ (M, Oi, Os) \text{ sat prop NONE} \]

\[ \text{[TRrule0]} \]
\[ \vdash \text{TR} (M, Oi, Os) \ (\text{exec cmd}) \]
\[ (\text{CFG authenticationTest stateInterp securityContext} \]

10
(P says prop (SOME cmd)::ins) s outs)
(CFG authenticationTest stateInterp securityContext ins
 (NS s (exec cmd)) (Out s (exec cmd)::outs)) \iff authenticationTest (P says prop (SOME cmd)) ∧
CFGInterpret (M,Oi,Os)
(CFG authenticationTest stateInterp securityContext
 (P says prop (SOME cmd)::ins) s outs)

[TRrule1]
\[\vdash TR (M,Oi,Os) (trap cmd)
 (CFG authenticationTest stateInterp securityContext
 (P says prop (SOME cmd)::ins) s outs)
 (CFG authenticationTest stateInterp securityContext ins
 (NS s (trap cmd)) (Out s (trap cmd)::outs)) \iff authenticationTest (P says prop (SOME cmd)) ∧
CFGInterpret (M,Oi,Os)
 (CFG authenticationTest stateInterp securityContext
 (P says prop (SOME cmd)::ins) s outs)

[trType_distinct_clauses]
\[\vdash (\forall a' a. \text{discard } a \neq \text{trap } a') ∧ (\forall a' a. \text{discard } a \neq \text{exec } a') ∧
 (\forall a' a. \text{trap } a \neq \text{exec } a')

[trType_one_one]
\[\vdash (\forall a' a'. (\text{discard } a = \text{discard } a') \iff (a = a')) ∧
 (\forall a' a'. (\text{trap } a = \text{trap } a') \iff (a = a')) ∧
 (\forall a' a'. (\text{exec } a = \text{exec } a') \iff (a = a'))

3 ssm Theory

Built: 10 June 2018
Parent Theories: satList

3.1 Datatypes

configuration =
  CFG (('command option, 'principal, 'd, 'e) Form -> bool)
  ('state ->
    ('command option, 'principal, 'd, 'e) Form list ->
    ('command option, 'principal, 'd, 'e) Form list)
  (('command option, 'principal, 'd, 'e) Form list ->
    ('command option, 'principal, 'd, 'e) Form list)
  (('command option, 'principal, 'd, 'e) Form list list)
  'state ('output list)

trType = discard 'cmdlist | trap 'cmdlist | exec 'cmdlist
3.2 Definitions

[authenticationTest_def]
\[ \forall \text{elementTest } x. \]
\[ \text{authenticationTest } \text{elementTest } x \iff \]
\[ \text{FOLDR } (\lambda p q. p \land q) \text{T } \text{MAP } \text{elementTest } x \]

[commandList_def]
\[ \forall x. \text{commandList } x = \text{MAP } \text{extractCommand } x \]

[inputList_def]
\[ \forall xs. \text{inputList } xs = \text{MAP } \text{extractInput } xs \]

[propCommandList_def]
\[ \forall x. \text{propCommandList } x = \text{MAP } \text{extractPropCommand } x \]

[TR_def]
\[ \text{TR } = \]
\[ (\lambda a_0 a_1 a_2 a_3. \]
\[ \forall TR'. \]
\[ (\exists \text{elementTest } NS M Oi Os Out s context stateInterp } x \]
\[ \text{ins outs}. \]
\[ (a_0 = (M,Oi,Os)) \land (a_1 = \text{exec } \text{inputList } x)) \land \]
\[ (a_2 = \]
\[ \text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::ins) s \]
\[ \text{outs}) \land \]
\[ (a_3 = \]
\[ \text{CFG } \text{elementTest } \text{stateInterp } \text{context } \text{ins} \]
\[ (NS s (\text{exec } \text{inputList } x))) \land \]
\[ (Out s (\text{exec } \text{inputList } x)::outs)) \land \]
\[ \text{authenticationTest } \text{elementTest } x \land \]
\[ \text{CFGInterpret } (M,Oi,Os) \]
\[ (\text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::ins) s \]
\[ \text{outs})) \lor \]
\[ (\exists \text{elementTest } NS M Oi Os Out s context stateInterp } x \]
\[ \text{ins outs}. \]
\[ (a_0 = (M,Oi,Os)) \land (a_1 = \text{trap } \text{inputList } x)) \land \]
\[ (a_2 = \]
\[ \text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::ins) s \]
\[ \text{outs}) \land \]
\[ (a_3 = \]
\[ \text{CFG } \text{elementTest } \text{stateInterp } \text{context } \text{ins} \]
\[ (NS s (\text{trap } \text{inputList } x))) \land \]
\[ (Out s (\text{trap } \text{inputList } x)::outs)) \land \]
\[ \text{authenticationTest } \text{elementTest } x \land \]
\[ \text{CFGInterpret } (M,Oi,Os) \]
\[ (\text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::ins) s \]

\[ 12 \]
(\square \text{elementTest} \text{ NS } M \text{ Oi } \text{ Os } \text{ Out} \text{ s context stateInterp} \text{ x}
\text{ ins outs}.  
(a_0 = (M, Oi, Os)) \land (a_1 = \text{discard (inputList x)}) \land 
(a_2 = 
\text{CFG elementTest stateInterp context} (x::\text{ins}) \text{ s outs}) \land 
(a_3 = 
\text{CFG elementTest stateInterp context} \text{ ins}
\text{ (NS s (discard (inputList x))))} 
\land 
\text{authenticationTest elementTest x}) \Rightarrow 
TR^i a_0 a_1 a_2 a_3 \Rightarrow 
TR^i a_0 a_1 a_2 a_3)

3.3 Theorems

[CFGInterpret_def]
\vdash \text{CFGInterpret} (M, Oi, Os)
\quad (\text{CFG elementTest stateInterp context} (x::\text{ins}) \text{ state outStream}) \iff 
(M, Oi, Os) \text{ satList context} \text{ x} \land (M, Oi, Os) \text{ satList} \text{ x} \land 
(M, Oi, Os) \text{ satList stateInterp state} \text{ x}

[CFGInterpret_ind]
\vdash \forall P. 
\quad (\forall M Oi Os \text{ elementTest stateInterp context} \text{ x} \text{ ins state outStream}, 
\quad P (M, Oi, Os) 
\quad (\text{CFG elementTest stateInterp context} (x::\text{ins}) \text{ state outStream})) \land 
\quad (\forall v_{15} v_{10} v_{11} v_{12} v_{13} v_{14}. 
\quad P v_{15} (\text{CFG} v_{10} v_{11} v_{12} \square v_{13} v_{14})) \Rightarrow 
\forall v v_1 v_2 v_3. P (v, v_1, v_2) v_3

[configuration_one_one]
\vdash \forall a_0 a_1 a_2 a_3 a_4 a_5 a'_0 a'_1 a'_2 a'_3 a'_4 a'_5. 
\quad (\text{CFG} a_0 a_1 a_2 a_3 a_4 a_5 = \text{CFG} a'_0 a'_1 a'_2 a'_3 a'_4 a'_5) \iff 
\quad (a_0 = a'_0) \land (a_1 = a'_1) \land (a_2 = a'_2) \land (a_3 = a'_3) \land 
\quad (a_4 = a'_4) \land (a_5 = a'_5)

[extractCommand_def]
\vdash \text{extractCommand} (P \text{ says prop (SOME cmd)}) = \text{cmd}

[extractCommand_ind]
\vdash \forall P'. 
\quad (\forall P \text{ cmd}. P' (P \text{ says prop (SOME cmd)))) \land P' \text{ TT} \land P' \text{ FF} \land 
\quad (\forall v_1. P' (\text{prop} v_1)) \land (\forall v_3. P' (\text{notf} v_3)) \land
\((\forall v_6, v_7. \ P' (v_6 \text{ andf } v_7)) \land (\forall v_{10}, v_{11}. \ P' (v_{10} \text{ orf } v_{11})) \land (\forall v_{14}, v_{15}. \ P' (v_{14} \text{ imp } v_{15})) \land (\forall v_{18}, v_{19}. \ P' (v_{18} \text{ eqf } v_{19})) \land (\forall v_{129}. \ P' (v_{129} \text{ says } \text{TT})) \land (\forall v_{130}. \ P' (v_{130} \text{ says } \text{FF})) \land (\forall v_{132}. \ P' (v_{132} \text{ says prop NONE})) \land (\forall v_{133}, v_{66}. \ P' (v_{133} \text{ says } \text{notf } v_{66})) \land (\forall v_{134}, v_{69}, v_{70}. \ P' (v_{134} \text{ says } (v_{69} \text{ andf } v_{70}))) \land (\forall v_{135}, v_{73}, v_{74}. \ P' (v_{135} \text{ says } (v_{73} \text{ orf } v_{74}))) \land (\forall v_{136}, v_{77}, v_{78}. \ P' (v_{136} \text{ says } (v_{77} \text{ impf } v_{78}))) \land (\forall v_{137}, v_{81}, v_{82}. \ P' (v_{137} \text{ says } (v_{81} \text{ eqf } v_{82}))) \land (\forall v_{138}, v_{85}, v_{86}. \ P' (v_{138} \text{ says } v_{85} \text{ says } v_{86})) \land (\forall v_{139}, v_{89}, v_{90}. \ P' (v_{139} \text{ says } v_{89} \text{ speaks_for } v_{90}) \land (\forall v_{140}, v_{93}, v_{94}. \ P' (v_{140} \text{ says } v_{93} \text{ controls } v_{94})) \land (\forall v_{141}, v_{98}, v_{99}, v_{100}. \ P' (v_{141} \text{ says } \text{reps } v_{98} v_{99} v_{100})) \land (\forall v_{142}, v_{103}, v_{104}. \ P' (v_{142} \text{ says } v_{103} \text{ domi } v_{104})) \land (\forall v_{143}, v_{107}, v_{108}. \ P' (v_{143} \text{ says } v_{107} \text{ eqi } v_{108})) \land (\forall v_{144}, v_{111}, v_{112}. \ P' (v_{144} \text{ says } v_{111} \text{ doms } v_{112})) \land (\forall v_{145}, v_{115}, v_{116}. \ P' (v_{145} \text{ says } v_{115} \text{ eqs } v_{116})) \land (\forall v_{146}, v_{119}, v_{120}. \ P' (v_{146} \text{ says } v_{119} \text{ eqn } v_{120})) \land (\forall v_{147}, v_{123}, v_{124}. \ P' (v_{147} \text{ says } v_{123} \text{ lte } v_{124})) \land (\forall v_{148}, v_{127}, v_{128}. \ P' (v_{148} \text{ says } v_{127} \text{ lte } v_{128})) \land (\forall v_{24}, v_{25}. \ P' (v_{24} \text{ speaks_for } v_{25})) \land (\forall v_{28}, v_{29}. \ P' (v_{28} \text{ controls } v_{29})) \land (\forall v_{33}, v_{34}, v_{35}. \ P' (\text{reps } v_{33} v_{34} v_{35})) \land (\forall v_{38}, v_{39}. \ P' (v_{38} \text{ domi } v_{39})) \land (\forall v_{42}, v_{43}. \ P' (v_{42} \text{ eqi } v_{43})) \land (\forall v_{46}, v_{47}. \ P' (v_{46} \text{ doms } v_{47})) \land (\forall v_{50}, v_{51}. \ P' (v_{50} \text{ eqs } v_{51})) \land (\forall v_{54}, v_{55}. \ P' (v_{54} \text{ eqn } v_{55})) \land (\forall v_{58}, v_{59}. \ P' (v_{58} \text{ lte } v_{59})) \land (\forall v_{62}, v_{63}. \ P' (v_{62} \text{ lte } v_{63})) \Rightarrow \forall v. \ P' v

[extractInput_def]
\vdash \text{extractInput } (P \text{ says prop } x) = x

[extractInput_ind]
\vdash \forall P'.
\begin{align*}
(\forall P x. \ P' (P \text{ says prop } x)) & \land P' \text{ TT } \land P' \text{ FF } \land \\
(\forall v_1. \ P' (\text{prop } v_1)) & \land (\forall v_3. \ P' (\text{notf } v_3)) \land \\
(\forall v_6, v_7. \ P' (v_6 \text{ andf } v_7)) & \land (\forall v_{10}, v_{11}. \ P' (v_{10} \text{ orf } v_{11})) \land \\
(\forall v_{14}, v_{15}. \ P' (v_{14} \text{ impf } v_{15})) & \land \\
(\forall v_{18}, v_{19}. \ P' (v_{18} \text{ eqf } v_{19})) & \land (\forall v_{129}. \ P' (v_{129} \text{ says } \text{TT})) \land \\
(\forall v_{130}. \ P' (v_{130} \text{ says } \text{FF})) & \land \\
(\forall v_{131}, v_{66}. \ P' (v_{131} \text{ says } \text{notf } v_{66})) & \land \\
(\forall v_{132}, v_{69}, v_{70}. \ P' (v_{132} \text{ says } (v_{69} \text{ andf } v_{70}))) & \land \\
(\forall v_{133}, v_{73}, v_{74}. \ P' (v_{133} \text{ says } (v_{73} \text{ orf } v_{74}))) & \land \\
(\forall v_{134}, v_{77}, v_{78}. \ P' (v_{134} \text{ says } (v_{77} \text{ impf } v_{78}))) & \land \\
(\forall v_{135}, v_{81}, v_{82}. \ P' (v_{135} \text{ says } (v_{81} \text{ eqf } v_{82}))) & \land
\end{align*}
\begin{align*}
&\forall v_{136} v_{85} v_{86}. \ P' (v_{136} \ \text{says} \ v_{85} \ \text{says} \ v_{86}) \land \\
&\forall v_{137} v_{89} v_{90}. \ P' (v_{137} \ \text{says} \ v_{89} \ \text{speaks_for} \ v_{90}) \land \\
&\forall v_{138} v_{93} v_{94}. \ P' (v_{138} \ \text{says} \ v_{93} \ \text{controls} \ v_{94}) \land \\
&\forall v_{139} v_{98} v_{99} v_{100}. \ P' (v_{139} \ \text{says} \ v_{98} \ v_{99} \ v_{100}) \land \\
&\forall v_{140} v_{103} v_{104}. \ P' (v_{140} \ \text{says} \ v_{103} \ \text{domi} \ v_{104}) \land \\
&\forall v_{141} v_{107} v_{108}. \ P' (v_{141} \ \text{says} \ v_{107} \ \text{eqi} \ v_{108}) \land \\
&\forall v_{142} v_{111} v_{112}. \ P' (v_{142} \ \text{says} \ v_{111} \ \text{doms} \ v_{112}) \land \\
&\forall v_{143} v_{115} v_{116}. \ P' (v_{143} \ \text{says} \ v_{115} \ \text{eqs} \ v_{116}) \land \\
&\forall v_{144} v_{119} v_{120}. \ P' (v_{144} \ \text{says} \ v_{119} \ \text{eqn} \ v_{120}) \land \\
&\forall v_{145} v_{123} v_{124}. \ P' (v_{145} \ \text{says} \ v_{123} \ \text{lte} \ v_{124}) \land \\
&\forall v_{146} v_{127} v_{128}. \ P' (v_{146} \ \text{says} \ v_{127} \ \text{lt} \ v_{128}) \land \\
&\forall v_{124} v_{25}. \ P' (v_{24} \ \text{speaks_for} \ v_{25}) \land \\
&\forall v_{128} v_{29}. \ P' (v_{28} \ \text{controls} \ v_{29}) \land \\
&\forall v_{133} v_{34} v_{35}. \ P' (\text{reps} v_{33} v_{34} v_{35}) \land \\
&\forall v_{138} v_{39}. \ P' (v_{138} \ \text{domi} \ v_{39}) \land \\
&\forall v_{142} v_{43}. \ P' (v_{142} \ \text{eqi} \ v_{43}) \land \\
&\forall v_{146} v_{47}. \ P' (v_{146} \ \text{doms} \ v_{47}) \land \\
&\forall v_{150} v_{51}. \ P' (v_{150} \ \text{eqs} \ v_{51}) \land \\
&\forall v_{154} v_{55}. \ P' (v_{154} \ \text{eqn} \ v_{55}) \land \\
&\forall v_{158} v_{59}. \ P' (v_{158} \ \text{lte} \ v_{59}) \land \\
&\forall v_{162} v_{63}. \ P' (v_{162} \ \text{lt} \ v_{63}) \Rightarrow \\
&\forall v. \ P' v
\end{align*}
SSM THEORY

Theorems

$$\forall v_{147} v_{123} v_{124}. P' (v_{147} \text{ says } v_{123} \text{ lte } v_{124}) \land$$
$$\forall v_{148} v_{127} v_{128}. P' (v_{148} \text{ says } v_{127} \text{ lt } v_{128}) \land$$
$$\forall v_{24} v_{25}. P' (v_{24} \text{ speaks_for } v_{25}) \land$$
$$\forall v_{28} v_{29}. P' (v_{28} \text{ controls } v_{29}) \land$$
$$\forall v_{43} v_{35}. P' (v_{43} \text{ eqn } v_{35}) \land$$
$$\forall v_{46} v_{47}. P' (v_{46} \text{ doms } v_{47}) \land$$
$$\forall v_{50} v_{51}. P' (v_{50} \text{ eqs } v_{51}) \land$$
$$\forall v_{54} v_{55}. P' (v_{54} \text{ eqn } v_{55}) \land$$
$$\forall v_{58} v_{59}. P' (v_{58} \text{ lte } v_{59}) \land$$
$$\forall v. \ P' v$$

[TR_cases]

\[ \forall a_0 a_1 a_2 a_3. \]

\[ \triangleright \forall a_0 a_1 a_2 a_3 \iff \]

\[ \exists \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. \]

\[ (a_0 = (M,Oi,Os)) \land (a_1 = \text{exec (inputList x)}) \land \]

\[ (a_2 = \text{CFG elementTest stateInterp context (x::ins) s outs}) \land \]

\[ (a_3 = \text{CFG elementTest stateInterp context ins}
\]

\[ (NS s (\text{exec (inputList x)})) \land \]

\[ (\text{Out s (exec (inputList x))::outs})) \land \]

\[ \text{authenticationTest elementTest x} \land \]

\[ \text{CFGInterpret (M,Oi,Os)} \]

\[ (\text{CFG elementTest stateInterp context (x::ins) s outs})) \lor \]

\[ \exists \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. \]

\[ (a_0 = (M,Oi,Os)) \land (a_1 = \text{trap (inputList x)}) \land \]

\[ (a_2 = \text{CFG elementTest stateInterp context (x::ins) s outs}) \land \]

\[ (a_3 = \text{CFG elementTest stateInterp context ins}
\]

\[ (NS s (\text{trap (inputList x)})) \land \]

\[ (\text{Out s (trap (inputList x))::outs})) \land \]

\[ \text{authenticationTest elementTest x} \land \]

\[ \text{CFGInterpret (M,Oi,Os)} \]

\[ (\text{CFG elementTest stateInterp context (x::ins) s outs})) \lor \]

\[ \exists \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. \]

\[ (a_0 = (M,Oi,Os)) \land (a_1 = \text{discard (inputList x)}) \land \]

\[ (a_2 = \text{CFG elementTest stateInterp context (x::ins) s outs}) \land \]

\[ (a_3 = \text{CFG elementTest stateInterp context (x::ins) s outs}) \lor \]

16
Theorems

[TR_discard_cmd_rule]
\[ \vdash \text{TR} (M, O_i, O_s) \ (\text{discard} \ (\text{inputList} \ x)) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ ins \]
\[ (NS \ s \ (\text{discard} \ (\text{inputList} \ x))) \]
\[ (Out \ s \ (\text{discard} \ (\text{inputList} \ x))::\text{outs})) \iff \neg \text{authenticationTest} \ \text{elementTest} \ x \]

[TR_EQ_rules_thm]
\[ \vdash (\text{TR} (M, O_i, O_s) \ (\text{exec} \ (\text{inputList} \ x)) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ ins \]
\[ (NS \ s \ (\text{exec} \ (\text{inputList} \ x))) \]
\[ (Out \ s \ (\text{exec} \ (\text{inputList} \ x))::\text{outs})) \iff \neg \text{authenticationTest} \ \text{elementTest} \ x \]
\[ \text{CFGInterpret} \ (M, O_i, O_s) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs})) \]
\[ (\text{TR} (M, O_i, O_s) \ (\text{trap} \ (\text{inputList} \ x)) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ ins \]
\[ (NS \ s \ (\text{trap} \ (\text{inputList} \ x))) \]
\[ (Out \ s \ (\text{trap} \ (\text{inputList} \ x))::\text{outs}) \iff \neg \text{authenticationTest} \ \text{elementTest} \ x \]
\[ \text{CFGInterpret} \ (M, O_i, O_s) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs})) \]
\[ (\text{TR} (M, O_i, O_s) \ (\text{discard} \ (\text{inputList} \ x)) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ ins \]
\[ (NS \ s \ (\text{discard} \ (\text{inputList} \ x))) \]
\[ (Out \ s \ (\text{discard} \ (\text{inputList} \ x))::\text{outs}) \iff \neg \text{authenticationTest} \ \text{elementTest} \ x \]

[TR_exec_cmd_rule]
\[ \forall \text{elementTest} \ \text{context} \ \text{stateInterp} \ x \ \text{ins} \ s \ \text{outs}. \]
\[ \forall M \ O_i \ O_s. \]
\[ \text{CFGInterpret} \ (M, O_i, O_s) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \Rightarrow \]
\[ (M, O_i, O_s) \ \text{satList} \ \text{propCommandList} \ x \Rightarrow \]
\[ \forall NS \ Out \ M \ O_i \ O_s. \]
\[ \text{TR} (M, O_i, O_s) \ (\text{exec} \ (\text{inputList} \ x)) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \]
\[ (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ ins \]
\[(NS\ s\ \text{exec}\ (inputList\ x))\]  
\[(Out\ s\ \text{exec}\ (inputList\ x))::\text{outs})\]  
\[\iff\]  
\[\text{authenticationTest}\ \text{elementTest}\ x\ \land\ \]  
\[\text{CFGInterpret}\ (M, O_i, O_s)\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\land\ (M, O_i, O_s)\ \text{satList}\ \text{propCommandList}\ x\]  

[\textbf{TR\_ind}]  
\[\vdash\forall\ TR'.\]  
\[\forall\ \text{elementTest}\ N S M O_i O_s Out\ s\ \text{context}\ \text{stateInterp}\ x\ \text{ins}\ \text{outs}.\]  
\[\text{authenticationTest}\ \text{elementTest}\ x\ \land\ \]  
\[\text{CFGInterpret}\ (M, O_i, O_s)\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\implies\]  
\[TR'\ (M, O_i, O_s)\ \text{exec}\ (inputList\ x))\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\text{CFG elementTest stateInterp context}\ \text{ins}\]  
\[\text{(NS\ s\ \text{exec}\ (inputList\ x)))}\]  
\[\text{(Out\ s\ (exec\ (inputList\ x)))::\text{outs})})\ \land\]  
\[\forall\ \text{elementTest}\ N S M O_i O_s Out\ s\ \text{context}\ \text{stateInterp}\ x\ \text{ins}\ \text{outs}.\]  
\[\text{authenticationTest}\ \text{elementTest}\ x\ \land\ \]  
\[\text{CFGInterpret}\ (M, O_i, O_s)\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\implies\]  
\[TR'\ (M, O_i, O_s)\ \text{trap}\ (inputList\ x))\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\text{CFG elementTest stateInterp context}\ \text{ins}\]  
\[\text{(NS\ s\ (trap\ (inputList\ x)))}\]  
\[\text{(Out\ s\ (trap\ (inputList\ x)))::\text{outs})})\ \land\]  
\[\forall\ \text{elementTest}\ N S M O_i O_s Out\ s\ \text{context}\ \text{stateInterp}\ x\ \text{ins}\ \text{outs}.\]  
\[\neg\text{authenticationTest}\ \text{elementTest}\ x\ \implies\]  
\[TR'\ (M, O_i, O_s)\ \text{discard}\ (inputList\ x))\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\text{CFG elementTest stateInterp context}\ \text{ins}\]  
\[\text{(NS\ s\ (discard\ (inputList\ x)))}\]  
\[\text{(Out\ s\ (discard\ (inputList\ x)))::\text{outs})})\ \Rightarrow\]  
\[\forall\ a_0\ a_1\ a_2\ a_3.\ TR\ a_0\ a_1\ a_2\ a_3\ \Rightarrow\ TR'\ a_0\ a_1\ a_2\ a_3\]  

[\textbf{TR\_rules}]  
\[\vdash\forall\ \text{elementTest}\ N S M O_i O_s Out\ s\ \text{context}\ \text{stateInterp}\ x\ \text{ins}\ \text{outs}.\]  
\[\text{authenticationTest}\ \text{elementTest}\ x\ \land\ \]  
\[\text{CFGInterpret}\ (M, O_i, O_s)\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  
\[\implies\]  
\[\text{TR}\ (M, O_i, O_s)\ \text{exec}\ (inputList\ x))\]  
\[\text{CFG elementTest stateInterp context}\ (x::\text{ins})\ s\ \text{outs}\]  

18
\[
\begin{align*}
\text{(CFG elementTest stateInterp context ins} & \quad \text{(NS s (exec (inputList x))}) \\
& \quad \text{(Out s (exec (inputList x))::outs))} \land \\
\forall \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. & \\
\text{authenticationTest elementTest x} \land \\
\text{CFGInterpret (M,Oi,Os)} & \Rightarrow \\
\text{(CFG elementTest stateInterp context (x::ins) s outs}) \Rightarrow & \\
\text{TR (M,Oi,Os) (exec (inputList x))} & \\
\text{(CFG elementTest stateInterp context (x::ins) s outs}) & \\
\text{(CFG elementTest stateInterp context ins} & \\
\text{TR (M,Oi,Os) (exec (inputList x))} & \\
\text{TR (M,Oi,Os) (trap (inputList x))} & \\
\forall \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. & \\
\neg \text{authenticationTest elementTest x} \Rightarrow & \\
\text{TR (M,Oi,Os) (discard (inputList x))} & \\
\forall \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. & \\
\text{authenticationTest elementTest x} \land \\
\text{CFGInterpret (M,Oi,Os)} & \Rightarrow \\
\text{(CFG elementTest stateInterp context (x::ins) s outs} & \\
\text{TR (M,Oi,Os) (exec (inputList x))} & \\
\text{(CFG elementTest stateInterp context (x::ins) s outs}) & \\
\text{(CFG elementTest stateInterp context ins} & \\
\text{TR (M,Oi,Os) (trap (inputList x))} & \\
\forall \text{elementTest NS M Oi Os Out s context stateInterp x ins outs}. & \\
\neg \text{authenticationTest elementTest x} \Rightarrow & \\
\text{TR (M,Oi,Os) (discard (inputList x))}
\end{align*}
\]
(CFG elementTest stateInterp context (x::ins) s outs)
(CFG elementTest stateInterp context ins
 (NS s (discard (inputList x)))
 (Out s (discard (inputList x))::outs))) ⇒
∀ a₀ a₁ a₂ a₃, TR a₀ a₁ a₂ a₃ ⇒ TR' a₀ a₁ a₂ a₃

[TR_trap_cmd_rule]
⊢ ∀ elementTest context stateInterp x ins s outs.
∀ M Oi Os.
CFGInterpret (M,Oi,Os)
 (CFG elementTest stateInterp context (x::ins) s
 outs) ⇒
 (M,Oi,Os) sat prop NONE ⇒
∀ NS Out M Oi Os.
TR (M,Oi,Os) (trap (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
 (NS s (trap (inputList x)))
 (Out s (trap (inputList x))::outs)) ⇔
authenticationTest elementTest x ∧
CFGInterpret (M,Oi,Os)
 (CFG elementTest stateInterp context (x::ins) s outs) ∧
 (M,Oi,Os) sat prop NONE

[TRrule0]
⊢ TR (M,Oi,Os) (exec (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
 (NS s (exec (inputList x)))
 (Out s (exec (inputList x))::outs)) ⇔
authenticationTest elementTest x ∧
CFGInterpret (M,Oi,Os)
 (CFG elementTest stateInterp context (x::ins) s outs)

[TRrule1]
⊢ TR (M,Oi,Os) (trap (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
 (NS s (trap (inputList x)))
 (Out s (trap (inputList x))::outs)) ⇔
authenticationTest elementTest x ∧
CFGInterpret (M,Oi,Os)
 (CFG elementTest stateInterp context (x::ins) s outs)

[trType_distinct_clauses]
⊢ (∀ a' a. discard a ≠ trap a') ∧ (∀ a' a. discard a ≠ exec a') ∧
∀ a' a. trap a ≠ exec a'
4 satList Theory

Built: 10 June 2018
Parent Theories: aclDrules

4.1 Definitions

[satList_def]
\[ \forall M\ Oi\ Os\ formList.\]
\[ (M,Oi,Os)\ satList\ formList \iff\]
\[ \text{FOLDR}\ (\lambda x\ y.\ x \land y)\ T\ (\text{MAP}\ (\lambda f.\ (M,Oi,Os)\ sat\ f)\ formList) \]

4.2 Theorems

[satList_conj]
\[ \forall l_1\ l_2\ M\ Oi\ Os.\]
\[ (M,Oi,Os)\ satList\ l_1\ \land\ (M,Oi,Os)\ satList\ l_2 \iff\]
\[ (M,Oi,Os)\ sat\List\ (l_1\ ++\ l_2) \]

[satList_CONS]
\[ \forall h\ t\ M\ Oi\ Os.\]
\[ (M,Oi,Os)\ satList\ (h::t) \iff\]
\[ (M,Oi,Os)\ sat\ h\ \land\ (M,Oi,Os)\ satList\ t \]

[satList_nil]
\[ \forall (M,Oi,Os)\ satList\ [] \]

5 PBTypeIntegrated Theory

Built: 11 June 2018
Parent Theories: OMNIType

5.1 Datatypes

\[ \text{omniCommand} = \text{ssmPlanPBCOMPLETE} | \text{ssmMoveToORPCOMPLETE} \]
\[ | \text{ssmConductORPCOMPLETE} | \text{ssmMoveToPBCOMPLETE} \]
\[ | \text{ssmConductPBCOMPLETE} | \text{invalidOmniCommand} \]

\[ \text{plCommand} = \text{crossLD} | \text{conductORP} | \text{moveToPB} | \text{conductPB} \]
\[ | \text{completePB} | \text{incomplete} \]
slCommand =
    PL PBTypeIntegrated$plCommand
    | OMNI PBTypeIntegrated$omniCommand

slOutput = PlanPB | MoveToORP | ConductORP | MoveToPB
    | ConductPB | CompletePB | unAuthenticated
    | unAuthorized

slState = PLAN_PB | MOVE_TO_ORP | CONDUCT_ORP | MOVE_TO_PB
    | CONDUCT_PB | COMPLETE_PB

stateRole = PlatoonLeader | Omni

5.2 Theorems

[omniCommand_distinct_clauses]
\[\vdash \text{ssmPlanPBComplete} \neq \text{ssmMoveToORPComplete} \land \]
\[\text{ssmPlanPBComplete} \neq \text{ssmConductORPComplete} \land \]
\[\text{ssmPlanPBComplete} \neq \text{ssmMoveToPBComplete} \land \]
\[\text{ssmPlanPBComplete} \neq \text{ssmConductPBComplete} \land \]
\[\text{ssmPlanPBComplete} \neq \text{invalidOmniCommand} \land \]
\[\text{ssmMoveToORPComplete} \neq \text{ssmConductORPComplete} \land \]
\[\text{ssmMoveToORPComplete} \neq \text{ssmMoveToPBComplete} \land \]
\[\text{ssmMoveToORPComplete} \neq \text{ssmConductPBComplete} \land \]
\[\text{ssmMoveToORPComplete} \neq \text{invalidOmniCommand} \land \]
\[\text{ssmConductORPComplete} \neq \text{ssmMoveToPBComplete} \land \]
\[\text{ssmConductORPComplete} \neq \text{ssmConductPBComplete} \land \]
\[\text{ssmConductORPComplete} \neq \text{invalidOmniCommand} \land \]
\[\text{ssmMoveToPBComplete} \neq \text{ssmConductPBComplete} \land \]
\[\text{ssmMoveToPBComplete} \neq \text{invalidOmniCommand} \land \]
\[\text{ssmConductPBComplete} \neq \text{invalidOmniCommand} \land \]

[plCommand_distinct_clauses]
\[\vdash \text{crossLD} \neq \text{conductORP} \land \text{crossLD} \neq \text{moveToPB} \land \]
\[\text{crossLD} \neq \text{conductPB} \land \text{crossLD} \neq \text{completePB} \land \]
\[\text{crossLD} \neq \text{incomplete} \land \text{conductORP} \neq \text{moveToPB} \land \]
\[\text{conductORP} \neq \text{conductPB} \land \text{conductORP} \neq \text{completePB} \land \]
\[\text{conductORP} \neq \text{incomplete} \land \text{moveToPB} \neq \text{conductPB} \land \]
\[\text{moveToPB} \neq \text{completePB} \land \text{moveToPB} \neq \text{incomplete} \land \]
\[\text{conductPB} \neq \text{completePB} \land \text{conductPB} \neq \text{incomplete} \land \]
\[\text{completePB} \neq \text{incomplete} \]

[slCommand_distinct_clauses]
\[\vdash \forall a' . \text{PL} a \neq \text{OMNI} a' \]

[slCommand_one_one]
\[\vdash (\forall a' . (\text{PL} a = \text{PL} a')) \iff (a = a') \land \]
\[\forall a a'. (\text{OMNI} a = \text{OMNI} a') \iff (a = a') \]
\[s1Output\_distinct\_clauses\]
\[s1State\_distinct\_clauses\]
\[stateRole\_distinct\_clauses\]

6 PBIntegratedDef Theory

Built: 11 June 2018
Parent Theories: PBTypeIntegrated, aclfoundation

6.1 Definitions

[secAuthorization\_def]
\(\forall x.\ secAuthorization\ x = secHelper\ (getOmniCommand\ x)\)

[secContext\_def]
\(\forall x.\ secContext\ PLAN\_PB\ x =\)
if getOmniCommand\ x = ssmPlanPBComplete then
   [prop\ (SOME\ (SLc\ (OMNI\ ssmPlanPBComplete)))\ impf
   Name PlatoonLeader controls
   prop\ (SOME\ (SLc\ (PL\ crossLD)))]]
```plaintext
else [prop NONE]) \land
(\forall xs.
  secContext MOVE_TO_ORP xs =
  if getOmniCommand xs = ssmMoveToORPComplete then
    [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL conductORP)))]
  else [prop NONE]) \land
(\forall xs.
  secContext CONDUCT_ORP xs =
  if getOmniCommand xs = ssmConductORPComplete then
    [prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL moveToPB)))]
  else [prop NONE]) \land
(\forall xs.
  secContext MOVE_TO_PB xs =
  if getOmniCommand xs = ssmMoveToPBComplete then
    [prop (SOME (SLc (OMNI ssmMoveToPBComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL conductPB)))]
  else [prop NONE]) \land
(\forall xs.
  secContext CONDUCT_PB xs =
  if getOmniCommand xs = ssmConductPBComplete then
    [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL completePB)))]
  else [prop NONE]

[secHelper_def]
\vdash \forall cmd.
  secHelper cmd =
  [Name Omni controls prop (SOME (SLc (OMNI cmd)))]

6.2 Theorems

[getOmniCommand_def]
\vdash (getOmniCommand [] = invalidOmniCommand) \land
(\forall xs cmd.
  getOmniCommand
  (Name Omni says prop (SOME (SLc (OMNI cmd))):xs) =
  cmd) \land
(\forall xs. getOmniCommand (TT::xs) = getOmniCommand xs) \land
(\forall xs. getOmniCommand (FF::xs) = getOmniCommand xs) \land
(\forall xs v_2. getOmniCommand (prop v_2::xs) = getOmniCommand xs) \land
(\forall xs v_3. getOmniCommand (notf v_3::xs) = getOmniCommand xs) \land
(\forall xs v_5 v_4.

24
getOmniCommand \(\forall z\; v_4\) andf \(v_5::xs\) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_6\) orf \(v_7::xs\) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_8\) impf \(v_9::xs\) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{10}\) eqf \(v_{11::xs}\) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{12}\) says TT::\(xs\) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{154}\). getOmniCommand (Name \(v_{134}\) says prop NONE::\(xs\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{144}\). getOmniCommand (Name PlatoonLeader says prop (SOME \(v_{144}\))::\(xs\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{146}\). getOmniCommand (Name Omni says prop (SOME (ESCc \(v_{146}\)))::\(xs\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{150}\). getOmniCommand (Name Omni says prop (SOME (SLc (PL \(v_{150}\))))::\(xs\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{68}\; v_{136}\; v_{135}\). getOmniCommand (v_{135} meet v_{136} says prop \(v_{68::xs}\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{68}\; v_{138}\; v_{137}\). getOmniCommand (v_{137} quoting v_{138} says prop \(v_{68::xs}\)) = getOmniCommand \(x\) \land

getOmniCommand \(\forall z\; v_{69}\; v_{12}\). getOmniCommand \(\forall z\; v_{71}\; v_{70}\; v_{12}\). getOmniCommand \(\forall z\; v_{73}\; v_{72}\; v_{12}\). getOmniCommand \(\forall z\; v_{75}\; v_{74}\; v_{12}\). getOmniCommand \(\forall z\; v_{77}\; v_{76}\; v_{12}\). getOmniCommand \(\forall z\; v_{79}\; v_{78}\; v_{12}\). getOmniCommand \(\forall z\; v_{12}\) says \(v_{78}\) says \(v_{79::xs}\) =
getOmniCommand $xs \land$

$(\forall \, \text{xs} \, \text{v}_{80} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{80}$ speaks_for $\text{v}_{81}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{82} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{82}$ controls $\text{v}_{83}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{84} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says reps $\text{v}_{84}$ $\text{v}_{85}$ $\text{v}_{86}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{87} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{87}$ domi $\text{v}_{88}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{89} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{89}$ eqi $\text{v}_{90}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{91} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{91}$ doms $\text{v}_{92}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{93} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{93}$ eqs $\text{v}_{94}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{95} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{95}$ eqn $\text{v}_{96}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{97} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{97}$ lte $\text{v}_{98}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{100} \, \text{v}_{12})$

getOmniCommand ($\text{v}_{12}$ says $\text{v}_{100}$ lt $\text{v}_{101}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{13} \, \text{v}_{14})$

getOmniCommand ($\text{v}_{14}$ speaks_for $\text{v}_{15}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{16} \, \text{v}_{17})$

getOmniCommand ($\text{v}_{16}$ controls $\text{v}_{17}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{19} \, \text{v}_{18})$

getOmniCommand (reps $\text{v}_{18}$ $\text{v}_{19}$ $\text{v}_{20}$::$\text{xs}$) =

getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{21} \, \text{v}_{22})$

getOmniCommand ($\text{v}_{21}$ domi $\text{v}_{22}$::$\text{xs}$) = getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{24} \, \text{v}_{23})$

getOmniCommand ($\text{v}_{23}$ eqi $\text{v}_{24}$::$\text{xs}$) = getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{26} \, \text{v}_{25})$

getOmniCommand ($\text{v}_{25}$ doms $\text{v}_{26}$::$\text{xs}$) = getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{28} \, \text{v}_{27})$

getOmniCommand ($\text{v}_{27}$ eqs $\text{v}_{28}$::$\text{xs}$) = getOmniCommand $\text{xs} \land$

$(\forall \, \text{xs} \, \text{v}_{30} \, \text{v}_{29})$

26
Theorems

\begin{align*}
\text{getOmniCommand} \ (v_{20} \ eqn \ v_{30}::xs) &= \text{getOmniCommand} \ xs \land \\
(\forall \ zs \ \exists s \ \exists t) \ &\land \\
\text{getOmniCommand} \ (v_{31} \ lte \ v_{32}::xs) &= \text{getOmniCommand} \ xs \land \\
(\forall \ zs \ \exists s \ \exists t) \ &\land \\
\text{getOmniCommand} \ (v_{31} \ lt \ v_{34}::xs) &= \text{getOmniCommand} \ xs
\end{align*}

\[\text{getOmniCommand\_ind}\]

\[
\vdash \ \forall P. \\
\quad P \ [] \land \\
(\forall cmd. \ xz) \ &\land \\
\quad P \ (\text{Name Omni says prop (SOME (SLc (OMNI cmd)))::xs}) \land \\
\quad (\forall xz. \ P xz \Rightarrow P (\text{TT::xs}) \land (\forall xz. \ P xz \Rightarrow P (\text{FF::xs}) \land \\
\quad (\forall v_{12} xz. \ P xz \Rightarrow P (\text{prop v}_{2}::xs)) \land \\
\quad (\forall v_{13} xz. \ P xz \Rightarrow P (\text{notf} v_{1}::xs)) \land \\
\quad (\forall v_{4} v_{5} xz. \ P xz \Rightarrow P (v_{4} \ \text{andf} \ v_{5}::xs)) \land \\
\quad (\forall v_{6} v_{7} xz. \ P xz \Rightarrow P (v_{6} \ \text{orf} \ v_{7}::xs)) \land \\
\quad (\forall v_{8} v_{9} xz. \ P xz \Rightarrow P (v_{8} \ \text{impf} \ v_{9}::xs)) \land \\
\quad (\forall v_{10} v_{11} xz. \ P xz \Rightarrow P (v_{10} \ \text{eqf} \ v_{11}::xs)) \land \\
\quad (\forall v_{12} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ \text{TT::xs})) \land \\
\quad (\forall v_{12} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ \text{FF::xs})) \land \\
\quad (\forall v_{134} xz. \ P xz \Rightarrow P (\text{Name v}_{134} \ \text{says prop NONE::xs})) \land \\
\quad (\forall v_{144} xz. \ P xz \Rightarrow P (\text{Name PlatoonLeader says prop (SOME v}_{144}::xs))) \land \\
\quad (\forall v_{146} xz. \ P xz \Rightarrow P (\text{Name Omni says prop (SOME (ESCc v}_{146})::xs))) \land \\
\quad (\forall v_{150} xz. \ P xz \Rightarrow P (v_{135} \ \text{meet} \ v_{136} \ \text{says prop v}_{68}::xs)) \land \\
\quad (\forall v_{137} v_{138} v_{68} xz. \ P xz \Rightarrow P (v_{137} \ \text{quoting} \ v_{138} \ \text{says prop v}_{68}::xs)) \land \\
\quad (\forall v_{12} v_{69} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ \text{notf} \ v_{69}::xs)) \land \\
\quad (\forall v_{12} v_{70} v_{71} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ (v_{70} \ \text{andf} \ v_{71})::xs)) \land \\
\quad (\forall v_{12} v_{72} v_{73} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ (v_{72} \ \text{orf} \ v_{73})::xs)) \land \\
\quad (\forall v_{12} v_{74} v_{75} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ (v_{74} \ \text{impf} \ v_{75})::xs)) \land \\
\quad (\forall v_{12} v_{76} v_{77} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ (v_{76} \ \text{eqf} \ v_{77})::xs)) \land \\
\quad (\forall v_{12} v_{78} v_{79} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{78} \ \text{says} v_{79}::xs)) \land \\
\quad (\forall v_{12} v_{80} v_{81} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{80} \ \text{speaks\_for} \ v_{81}::xs)) \land \\
\quad (\forall v_{12} v_{82} v_{83} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{82} \ \text{controls} \ v_{83}::xs)) \land \\
\quad (\forall v_{12} v_{84} v_{85} v_{86} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} \ rep\_f v_{84} v_{85} v_{86}::xs)) \land \\
\quad (\forall v_{12} v_{87} v_{88} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{87} \ \text{domi} v_{88}::xs)) \land \\
\quad (\forall v_{12} v_{89} v_{90} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{89} \ \text{eqi} v_{90}::xs)) \land \\
\quad (\forall v_{12} v_{91} v_{92} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{91} \ \text{doms} v_{92}::xs)) \land \\
\quad (\forall v_{12} v_{93} v_{94} xz. \ P xz \Rightarrow P (v_{12} \ \text{says} v_{93} \ \text{eqs} v_{94}::xs)) \land
\]
7 Theorems

(inputOK_cmd_reject_lemma)
\[ \vdash \forall \text{cmd}. \, \neg \text{inputOK} \,(\text{prop} \,(\text{SOME} \,(\text{cmd}))) \]

(inputOK_def)
\[ \vdash (\text{inputOK} \,(\text{Name} \, \text{PlatoonLeader} \, \text{says} \, \text{prop} \,(\text{cmd})) \iff T) \land \]
\[ (\text{inputOK} \,(\text{Name} \, \text{Omni} \, \text{says} \, \text{prop} \,(\text{cmd})) \iff T) \land \]
\[ (\text{inputOK} \,(\text{TT} \iff F) \land (\text{inputOK} \,(\text{FF} \iff F) \land \]
\[ (\text{inputOK} \,(\text{prop} \,(\text{v} \lnot \text{v})) \iff F) \land (\text{inputOK} \,(\text{notf} \,(\text{v})) \iff F) \land \]
\[ (\text{inputOK} \,(\text{v} \land \text{v}) \iff F) \land (\text{inputOK} \,(\text{orv} \,(\text{v})) \iff F) \land \]
\[ (\text{inputOK} \,(\text{impv} \,(\text{v})) \iff F) \land (\text{inputOK} \,(\text{eqv} \,(\text{v})) \iff F) \land \]

7 ssmPBIntegrated Theory

Built: 11 June 2018
Parent Theories: PBIintegratedDef, ssm
Theorems

SSMPBINTEGRATED THEORY

(inputOK (v10 says TT) \iff F) \land (inputOK (v10 says FF) \iff F) \land
(inputOK (v133 meet v134 says prop v96) \iff F) \land
(inputOK (v135 quoting v136 says prop v96) \iff F) \land
(inputOK (v10 says notf v87) \iff F) \land
(inputOK (v10 says (v68 andf v69)) \iff F) \land
(inputOK (v10 says (v70 orf v71)) \iff F) \land
(inputOK (v10 says (v72 impf v73)) \iff F) \land
(inputOK (v10 says (v74 eqf v75)) \iff F) \land
(inputOK (v10 says v76 says v77) \iff F) \land
(inputOK (v10 says v78 speaks_for v79) \iff F) \land
(inputOK (v10 says v80 controls v81) \iff F) \land
(inputOK (v10 says reps v82 v83 v84) \iff F) \land
(inputOK (v10 says v85 domi v86) \iff F) \land
(inputOK (v10 says v87 eqi v88) \iff F) \land
(inputOK (v10 says v89 doms v90) \iff F) \land
(inputOK (v10 says v91 eqs v92) \iff F) \land
(inputOK (v10 says v93 eqn v94) \iff F) \land
(inputOK (v10 says v95 lte v96) \iff F) \land
(inputOK (v10 says v97 lt v98) \iff F) \land
(inputOK (v12 speaks_for v13) \iff F) \land
(inputOK (v14 controls v15) \iff F) \land
(inputOK (reps v16 v17 v18) \iff F) \land
(inputOK (v19 domi v20) \iff F) \land
(inputOK (v21 eqi v22) \iff F) \land
(inputOK (v23 doms v24) \iff F) \land
(inputOK (v25 eqs v26) \iff F) \land (inputOK (v27 eqn v28) \iff F) \land
(inputOK (v29 lte v30) \iff F) \land (inputOK (v31 lt v32) \iff F)

[inputOK_ind]

\vdash \forall P.
\quad (\forall cmd. P (Name PlatoonLeader says prop cmd)) \land
\quad (\forall cmd. P (Name Omni says prop cmd)) \land P TT \land P FF \land
\quad (\forall v. P (prop v)) \land (\forall v1. P (notf v1)) \land
\quad (\forall v2 v3. P (v2 andf v3)) \land (\forall v4 v5. P (v4 orf v5)) \land
\quad (\forall v6 v7. P (v6 impf v7)) \land (\forall v8 v9. P (v8 eqf v9)) \land
\quad (\forall v10. P (v10 says TT)) \land (\forall v10. P (v10 says FF)) \land
\quad (\forall v133 v134 v96. P (v133 meet v134 says prop v96)) \land
\quad (\forall v135 v136 v96. P (v135 quoting v136 says prop v96)) \land
\quad (\forall v10 v87. P (v10 says notf v87)) \land
\quad (\forall v10 v68 v69. P (v10 says (v68 andf v69))) \land
\quad (\forall v10 v70 v71. P (v10 says (v70 orf v71))) \land
\quad (\forall v10 v72 v73. P (v10 says (v72 impf v73))) \land
\quad (\forall v10 v74 v75. P (v10 says (v74 eqf v75))) \land
\quad (\forall v10 v76 v77. P (v10 says v76 says v77)) \land
\quad (\forall v10 v78 v79. P (v10 says v78 speaks_for v79)) \land
\quad (\forall v10 v80 v81. P (v10 says v80 controls v81)) \land
\quad (\forall v10 v82 v83 v84. P (v10 says reps v82 v83 v84)) \land
\quad (\forall v10 v85 v86. P (v10 says v85 domi v86)) \land
\quad (\forall v10 v87 v88. P (v10 says v87 eqi v88)) \land

29
(∀ v_10 \ v_90. P (v_10 \ says \ v_90 \ doms \ v_90)) \land
(∀ v_10 \ v_91 \ v_92. P (v_10 \ says \ v_91 \ eqs \ v_92)) \land
(∀ v_10 \ v_93 \ v_94. P (v_10 \ says \ v_93 \ eqn \ v_94)) \land
(∀ v_10 \ v_95 \ v_96. P (v_10 \ says \ v_95 \ lte \ v_96)) \land
(∀ v_10 \ v_97 \ v_98. P (v_10 \ says \ v_97 \ lt \ v_98)) \land
(∀ v_{12} \ v_{13}. P (v_{12} \ speaks_for \ v_{13})) \land
(∀ v_{14} \ v_{15}. P (v_{14} \ controls \ v_{15})) \land
(∀ v_{16} \ v_{17} \ v_{18}. P (\reps \ v_{16} \ v_{17} \ v_{18})) \land
(∀ v_{21} \ v_{22}. P (v_{21} \ eqi \ v_{22})) \land
(∀ v_{23} \ v_{24}. P (v_{23} \ doms \ v_{24})) \land
(∀ v_{25} \ v_{26}. P (v_{25} \ eqs \ v_{26}) \land (\forall v_{27} v_{28}. P (v_{27} \ eqn \ v_{28}))) \land
(∀ v_{29} \ v_{30}. P (v_{29} \ lte \ v_{30}) \land (\forall v_{31} v_{32}. P (v_{31} \ lt \ v_{32}))) \Rightarrow
\forall v. P v.

[PBNS_def]

\vdash \text{(PBNS PLAN_PB (exec x) =}
\quad \text{if getPlCom x = crossLD then MOVE_TO_ORP else PLAN_PB) \land}
\text{(PBNS MOVE_TO_ORP (exec x) =}
\quad \text{if getPlCom x = conductORP then CONDUCT_ORP}
\quad \text{else MOVE_TO_ORP) \land}
\text{(PBNS CONDUCT_ORP (exec x) =}
\quad \text{if getPlCom x = moveToPB then MOVE_TO_PB else CONDUCT_ORP) \land}
\text{(PBNS MOVE_TO_PB (exec x) =}
\quad \text{if getPlCom x = conductPB then CONDUCT_PB else MOVE_TO_PB) \land}
\text{(PBNS CONDUCT_PB (exec x) =}
\quad \text{if getPlCom x = completePB then COMPLETE_PB}
\quad \text{else CONDUCT_PB) \land (PBNS s (trap v_0) = s) \land}
\text{(PBNS s (discard v_1) = s)}

[PBNS_ind]

\vdash \forall P.
\quad (\forall x. P \ PLAN_PB (exec x)) \land (\forall x. P \ MOVE_TO_ORP (exec x)) \land
\quad (\forall x. P \ CONDUCT_ORP (exec x)) \land
\quad (\forall x. P \ MOVE_TO_PB (exec x)) \land (\forall x. P \ CONDUCT_PB (exec x)) \land
\quad (\forall s v_0. P s (\trap v_0)) \land (\forall s v_1. P s (\discard v_1)) \land
\quad (\forall v_0. P \ COMPLETE_PB (exec v_0)) \Rightarrow
\forall v v_1. P v v_1.

[PBOut_def]

\vdash \text{(PBOut PLAN_PB (exec x) =}
\quad \text{if getPlCom x = crossLD then MoveToORP else PlanPB) \land}
\text{(PBOut MOVE_TO_ORP (exec x) =}
\quad \text{if getPlCom x = conductORP then ConductORP else MoveToORP) \land}
\text{(PBOut CONDUCT_ORP (exec x) =}
\quad \text{if getPlCom x = moveToPB then MoveToORP else ConductORP) \land}
\text{(PBOut MOVE_TO_PB (exec x) =}
\quad \text{if getPlCom x = conductPB then ConductPB else MoveToPB) \land}
(PBOut CONDUCT_PB (exec x) =
  if getPlCom x = completePB then CompletePB else ConductPB) ∧
(PBOut s (trap v₀) = unAuthorized) ∧
(PBOut s (discard v₁) = unAuthenticated)

[PBOut_ind]
⊢ ∀ P.
(∀ x. P PLAN_PB (exec x)) ∧ (∀ x. P MOVE_TO_ORP (exec x)) ∧
(∀ x. P CONDUCT_ORP (exec x)) ∧
(∀ x. P MOVE_TO_PB (exec x)) ∧ (∀ x. P CONDUCT_PB (exec x)) ∧
(∀ s v₀. P s (trap v₀)) ∧ (∀ v v₁. P s (discard v₁)) ∧
(∀ v v₀. P COMPLETE_PB (exec v₀)) ⇒
∀ v v₁. P v v₁

[PlatoonLeader_Omni_notDiscard_s1Command_thm]
⊢ ∀ NS Out M Oi Os.
¬TR (M,Oi,Os)
(discard
  [SOME (SLc (PL pCommand));
   SOME (SLc (OMNI omniCommand))]
(CFG inputOK secContext secAuthorization
  ([Name Omni says prop (SOME (SLc (PL pCommand)));
    Name PlatoonLeader says
    prop (SOME (SLc (OMNI omniCommand)))]::ins) PLAN_PB
outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(discard
  [SOME (SLc (PL pCommand));
   SOME (SLc (OMNI omniCommand))])
(Out PLAN_PB
(discard
  [SOME (SLc (PL pCommand));
   SOME (SLc (OMNI omniCommand))])::outs))

[PlatoonLeader_PLAN_PB_exec_justified_lemma]
⊢ ∀ NS Out M Oi Os.
TR (M,Oi,Os)
(exec
  [inputList
    [Name Omni says
      prop (SOME (SLc (OMNI ssmPlanPBComplete)));
      Name PlatoonLeader says
      prop (SOME (SLc (PL crossLD)))]])
(CFG inputOK secContext secAuthorization
([Name Omni says
  prop (SOME (SLc (OMNI ssmPlanPBComplete)));
  Name PlatoonLeader says
  prop (SOME (SLc (OMNI ssmPlanPBComplete)))];
  Name PlatoonLeader says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\) :: \(\text{ins}\) PLAN_PB outs
(CFG inputOK secContext secAuthorization \(\text{ins}\)
\(\text{exec}\)
\(\text{inputList}\)
[Name Omni says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\);
Name PlatoonLeader says
prop \((\text{SLc (PL crossLD))})\)
]\]
\(\text{Out}\) PLAN_PB
\(\text{exec}\)
\(\text{inputList}\)
[Name Omni says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\);
Name PlatoonLeader says
prop \((\text{SLc (PL crossLD))})\)] :: \(\text{outs}\)
\(\text{authenticationTest}\) inputOK
[Name Omni says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\);
Name PlatoonLeader says
prop \((\text{SLc (PL crossLD))})\] \(\land\)
\(\text{CFGInterpret}\) \(\langle M, Oi, Os \rangle\)
\(\text{exec}\)
\(\text{inputList}\)
[Name Omni says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\);
Name PlatoonLeader says
prop \((\text{SLc (PL crossLD))})\)] :: \(\text{ins}\) PLAN_PB outs \(\land\)
\(\langle M, Oi, Os \rangle\) satList
propCommandList
[Name Omni says
prop \((\text{SLc (OMNI ssmPlanPBComplete))})\);
Name PlatoonLeader says prop \((\text{SLc (PL crossLD))})\)

\[\text{PlatoonLeader\_PLAN\_PB\_exec\_justified\_thm}\]
\(\vdash \forall NS\ Out\ M\ Oi\ Os.\)
\(\text{TR}\) \(\langle M, Oi, Os \rangle\)
\(\text{exec}\)
description
\(\text{CFG inputOK secContext secAuthorization}\)
\(\langle\text{exec}\rangle\)
\(\langle\text{inputList}\rangle\)
\(\langle\text{propCommandList}\rangle\)
\[
\text{\textbf{Theorems}}
\]

\[
\text{\textbf{SSMPBINTEGRATED THEORY}}
\]

\[
\text{\textbf{PlatoonLeader\_PLAN\_PB\_exec\_lemma}}
\]

\[
\forall M, O_i, O_s.
\]
\[
\text{CFGInterpret} (M, O_i, O_s)
\]
\[
\text{CFG inputOK secContext secAuthorization}
\]
\[
([\text{Name Omni says prop (SOME (SLc (OMNI ssmPlanPBComplete)))};
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))] :: outs}) \Rightarrow
\]
\[
(M, O_i, O_s) \text{ satList propCommandList}
\]
\[
([\text{Name Omni says prop (SOME (SLc (OMNI ssmPlanPBComplete)))};
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))]}
\]

\[
\text{\textbf{PlatoonLeader\_PLAN\_PB\_trap\_justified\_lemma}}
\]

\[
\forall omniCommand \neq \text{ssmPlanPBComplete} \Rightarrow
\]
\[
s = \text{PLAN\_PB} \Rightarrow
\]
\[
\forall NS \ Out M, O_i, O_s.
\]
\[
\text{TR} (M, O_i, O_s)
\]
\[
\text{trap (inputList}
\]
\[
([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD))))}])))
\]
\[
\text{CFG inputOK secContext secAuthorization}
\]
\[
([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});
\text{Name PlatoonLeader says prop (SOME (SLc (PL crossLD))))}])))
\]
SSMPBINTEGRATED THEORY

Theorems

Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))): ins PLAN_PB outs)

(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(trap
    (inputList
      [Name Omni says
       prop (SOME (SLc (OMNI omniCommand)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL crossLD)))]))
  (Out PLAN_PB
    (trap
      (inputList
        [Name Omni says
         prop (SOME (SLc (OMNI omniCommand)));
         Name PlatoonLeader says
         prop (SOME (SLc (PL crossLD)))]):
       outs)
      authenticationTest inputOK
      [Name Omni says prop (SOME (SLc (OMNI omniCommand)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL crossLD)))] ^
      CFGInterpret (M, Oi, Os)
      (CFG inputOK secContext secAuthorization
       ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
         Name PlatoonLeader says
         prop (SOME (SLc (PL crossLD)))]:
        ins) PLAN_PB
       outs) ^ (M, Oi, Os) sat prop NONE

[PlatoonLeader_PLAN_PB_trap_justified_thm]
\[ omniCommand \neq ssmPlanPBComplete \implies \\
(s = PLAN_PB) \implies \\
\forall NS Out M Oi Os.
   TR (M, Oi, Os)
   (trap
     [SOME (SLc (OMNI omniCommand));
      SOME (SLc (PL crossLD))])
   (CFG inputOK secContext secAuthorization
    ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
      Name PlatoonLeader says
      prop (SOME (SLc (PL crossLD)))]:
     ins) PLAN_PB outs)
   (CFG inputOK secContext secAuthorization ins
    (NS PLAN_PB
     (trap
      [SOME (SLc (OMNI omniCommand));
       SOME (SLc (PL crossLD))])
    (Out PLAN_PB
     (trap
      [SOME (SLc (OMNI omniCommand));
       SOME (SLc (PL crossLD))]:
     outs)) \iff

34
authenticationTest inputOK

(Name Omni says prop (SOME (SLc (OMNI omniCommand))));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))] \东西
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
  ( [Name Omni says prop (SOME (SLc (OMNI omniCommand))]);
  Name PlatoonLeader says
  prop (SOME (SLc (PL crossLD)))] :: ins) PLAN_PB
outs) \东西 (M, Oi, Os) sat prop NONE

[PlatoonLeader_PLAN_PB_trap_lemma]
\(\vdash\) omniCommand \neq\ smmPlanPBComplete \Rightarrow
(s = PLAN_PB) \Rightarrow
\forall M\ Oi\ Os.
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
  ( [Name Omni says prop (SOME (SLc (OMNI omniCommand))]);
  Name PlatoonLeader says
  prop (SOME (SLc (PL crossLD)))] :: ins) PLAN_PB
outs) \Rightarrow
(M, Oi, Os) sat prop NONE

8 ssmConductORP Theory

Built: 11 June 2018
Parent Theories: ConductORPDef

8.1 Theorems

[conductORPNS_def]
\(\vdash\) (conductORPNS CONDUCT_ORP (exec x) =
  if getPlCom x = secure then SECURE else CONDUCT_ORP) \&
  (conductORPNS SECURE (exec x) =
  if getPsCom x = actionsIn then ACTIONS_IN else SECURE) \&
  (conductORPNS ACTIONS_IN (exec x) =
  if getPlCom x = withdraw then WITHDRAW else ACTIONS_IN) \&
  (conductORPNS WITHDRAW (exec x) =
  if getPlCom x = complete then COMPLETE else WITHDRAW) \&
  (conductORPNS s (trap x) = s) \&
  (conductORPNS s (discard x) = s)

[conductORPNS_ind]
\(\vdash\) \forall P.
  (\forall x. P CONDUCT_ORP (exec x)) \& (\forall x. P SECURE (exec x)) \&
  (\forall x. P ACTIONS_IN (exec x)) \& (\forall x. P WITHDRAW (exec x)) \&
  (\forall s x. P s (trap x)) \& (\forall s x. P s (discard x)) \&
\((\forall v_5. \ P \ COMPLETE (\text{exec } v_5)) \Rightarrow \\
\forall v v_1. \ P v v_1\)

\[\text{conductORPOut_def}\]
\[\vdash (\text{conductORPOut } CONDUCT\_ORP (\text{exec } x) = \\
\quad \text{if getPlCom } x = \text{secure then Secure else ConductORP}) \land \\
(\text{conductORPOut SECURE (\text{exec } x) = \\
\quad \text{if getPsgCom } x = \text{actionsIn then ActionsIn else Secure}) \land \\
(\text{conductORPOut ACTIONS\_IN (\text{exec } x) = \\
\quad \text{if getPlCom } x = \text{withdraw then Withdraw else ActionsIn}) \land \\
(\text{conductORPOut WITHDRAW (\text{exec } x) = \\
\quad \text{if getPlCom } x = \text{complete then Complete else Withdraw}) \land \\
(\text{conductORPOut s (\text{trap } x) = unAuthorized}) \land \\
(\text{conductORPOut s (\text{discard } x) = unAuthenticated})\]

\[\text{conductORPOut_ind}\]
\[\vdash \forall P. \\
(\forall x. \ P \ CONDUCT\_ORP (\text{exec } x)) \land (\forall x. \ P \ SECURE (\text{exec } x)) \land \\
(\forall x. \ P \ ACTIONS\_IN (\text{exec } x)) \land (\forall x. \ P \ WITHDRAW (\text{exec } x)) \land \\
(\forall s v_5. \ P s (\text{trap } x)) \land (\forall s x. \ P s (\text{discard } x)) \land \\
(\forall v_5. \ P \ COMPLETE (\text{exec } v_5)) \Rightarrow \\
\forall v v_1. \ P v v_1\]

\[\text{inputOK\_cmd\_reject\_lemma}\]
\[\vdash \forall \text{cmd}. \ \neg \text{inputOK (prop (SOME cmd))}\]

\[\text{inputOK\_def}\]
\[\vdash (\text{inputOK (Name PlatoonLeader says prop cmd) } \iff \ T) \land \\
(\text{inputOK (Name PlatoonSergeant says prop cmd) } \iff \ T) \land \\
(\text{inputOK TT } \iff \ F) \land (\text{inputOK FF } \iff \ F) \land \\
(\text{inputOK (prop } v) \iff \ F) \land (\text{inputOK (notf } v_1) \iff \ F) \land \\
(\text{inputOK (notf } v_2 \landf v_3) \iff \ F) \land (\text{inputOK (notf } v_4 \lorf v_5) \iff \ F) \land \\
(\text{inputOK (v_6 impf v_7) } \iff \ F) \land (\text{inputOK (v_8 eqf v_9) } \iff \ F) \land \\
(\text{inputOK (v_10 says TT) } \iff \ F) \land (\text{inputOK (v_10 says FF) } \iff \ F) \land \\
(\text{inputOK (v_133 meet v_134 says prop v_66) } \iff \ F) \land \\
(\text{inputOK (v_135 quoting v_136 says prop v_66) } \iff \ F) \land \\
(\text{inputOK (v_10 says notf v_67) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_68 andf v_69) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_70 orf v_71) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_72 impf v_73) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_74 eqf v_75) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_76 says v_77) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_78 speaks_for v_79) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_80 controls v_81) } \iff \ F) \land \\
(\text{inputOK (v_10 says reps v_82 v_83 v_84) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_85 domi v_86) } \iff \ F) \land \\
(\text{inputOK (v_10 says v_87 eqi v_88) } \iff \ F) \land \]
(\text{inputOK} (v_{10} \text{ says } v_{99} \text{ doms } v_{99}) \iff F) \land
(\text{inputOK} (v_{10} \text{ says } v_{91} \text{ eqs } v_{92}) \iff F) \land
(\text{inputOK} (v_{10} \text{ says } v_{93} \text{ eqn } v_{94}) \iff F) \land
(\text{inputOK} (v_{10} \text{ says } v_{95} \text{ lte } v_{96}) \iff F) \land
(\text{inputOK} (v_{10} \text{ says } v_{97} \text{ lt } v_{98}) \iff F) \land
(\text{inputOK} (v_{12} \text{ speaks_for } v_{13}) \iff F) \land
(\text{inputOK} (v_{14} \text{ controls } v_{15}) \iff F) \land
(\text{inputOK} (\text{reps } v_{16} \ v_{17} \ v_{18}) \iff F) \land
(\text{inputOK} (v_{19} \text{ domi } v_{20}) \iff F) \land
(\text{inputOK} (v_{21} \text{ eqi } v_{22}) \iff F) \land
(\text{inputOK} (v_{23} \text{ doms } v_{241}) \iff F) \land
(\text{inputOK} (v_{25} \text{ eqs } v_{26}) \iff F) \land (\text{inputOK} (v_{27} \text{ eqn } v_{28}) \iff F) \land
(\text{inputOK} (v_{29} \text{ lte } v_{30}) \iff F) \land (\text{inputOK} (v_{31} \text{ lt } v_{32}) \iff F)

\text{[inputOK\_ind]}

\vdash \forall P.
(\forall \text{ cmd}. \ P (\text{Name PlatoonLeader says prop cmd})) \land
(\forall \text{ cmd}. \ P (\text{Name PlatoonSergeant says prop cmd})) \land
(\forall \text{ v}. \ P (\text{prop v}) \land (\forall v_1. \ P (\text{notf } v_1)) \land
(\forall v_2. \ P (v_2 \text{ andf } v_3)) \land (\forall v_4. \ P (v_4 \text{ orf } v_5)) \land
(\forall v_6. \ P (v_6 \text{ impf } v_7)) \land (\forall v_8. \ P (v_8 \text{ eqf } v_9)) \land
(\forall v_{10}. \ P (v_{10} \text{ says TT})) \land (\forall v_{10}. \ P (v_{10} \text{ says FF})) \land
(\forall v_{133}. \ P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{90})) \land
(\forall v_{135}. \ P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{90})) \land
(\forall v_{10}. \ P (v_{10} \text{ says notf } v_{97})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{97} \text{ andf } v_{98}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{97} \text{ orf } v_{71}))) \land
(\forall v_{12}. \ P (v_{12} \text{ impf } v_{73})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{74} \text{ eqf } v_{75}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{76} \text{ says } v_{77}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{80} \text{ controls } v_{81}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{82} \text{ eqs } v_{84}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{85} \text{ doms } v_{86}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{89} \text{ doms } v_{90}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{95} \text{ lte } v_{96}))) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{97} \text{ lt } v_{98}))) \land
(\forall v_{12}. \ P (v_{12} \text{ speaks_for } v_{13}))) \land
(\forall v_{14}. \ P (v_{14} \text{ controls } v_{15}))) \land
(\forall v_{16}. \ P (\text{reps } v_{16} \ v_{17} \ v_{18}))) \land
(\forall v_{19}. \ P (v_{19} \text{ domi } v_{20}))) \land
(\forall v_{21}. \ P (v_{21} \text{ eqi } v_{22}))) \land
(\forall v_{23}. \ P (v_{23} \text{ doms } v_{24}))) \land
(\forall v_{25}. \ P (v_{25} \text{ eqs } v_{26}))) \land (\forall v_{27}. \ P (v_{27} \text{ eqn } v_{28}))) \land
(\forall v_{29}. \ P (v_{29} \text{ lte } v_{30}))) \land (\forall v_{31}. \ P (v_{31} \text{ lt } v_{32}))) \Rightarrow
∀ v. P v

[PlatoonLeader_ACTIONS_IN_exec_justified_lemma]

∀ NS Out M Oi Os.
   TR (M, Oi, Os)
   (exec
     (inputList
      [Name Omni says
       prop (SOME (SLc (OMNI ssmActionsInComplete)));
      Name PlatoonLeader says
       prop (SOME (SLc (PL withdraw))))])
   (CFG inputOK secContext secAuthorization
    ([Name Omni says
      prop (SOME (SLc (OMNI ssmActionsInComplete)));
    Name PlatoonLeader says
      prop (SOME (SLc (PL withdraw)))]::ins) ACTIONS_IN
    outs)
   (CFG inputOK secContext secAuthorization ins
    (NS ACTIONS_IN
     (exec
      (inputList
       [Name Omni says
        prop
        (SOME (SLc (OMNI ssmActionsInComplete)));
        Name PlatoonLeader says
        prop (SOME (SLc (PL withdraw))))])
     (Out ACTIONS_IN
      (exec
       (inputList
        [Name Omni says
         prop
         (SOME (SLc (OMNI ssmActionsInComplete)));
        Name PlatoonLeader says
        prop (SOME (SLc (PL withdraw))))::
        outs)) ↔
      authenticationTest inputOK
      [Name Omni says
       prop (SOME (SLc (OMNI ssmActionsInComplete)));
      Name PlatoonLeader says
       prop (SOME (SLc (PL withdraw)))] ∧
      CFGInterpret (M, Oi, Os)
      (CFG inputOK secContext secAuthorization
       ([Name Omni says
         prop (SOME (SLc (OMNI ssmActionsInComplete)));
       Name PlatoonLeader says
         prop (SOME (SLc (PL withdraw)))]::ins) ACTIONS_IN
       outs) ∧
      (M, Oi, Os) satList
      propCommandList

38
Theorems

[Name Omni says
prop (SOME (SLc (OMNI ssmActionsInComplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]

[PlatoonLeader_ACTIONS_IN_exec_justified_thm]
\[\forall NS \ Out \ M \ Oi \ Os.\]
\[\text{TR} \ (M, Oi, Os)\]
\[\text{(exec}\]
\[\text{[SOME (SLc (OMNI ssmActionsInComplete))];}\n\[\text{SOME (SLc (PL withdraw))]}\]
\[\text{(CFG inputOK secContext secAuthorization}\]
\[\text{([Name Omni says}\]
\[\text{prop (SOME (SLc (OMNI ssmActionsInComplete))));}\n\[\text{Name PlatoonLeader says}\]
\[\text{prop (SOME (SLc (PL withdraw)))); : ins} \text{ ACTIONS_IN}\]
\[\text{outs})}\]
\[\text{(CFG inputOK secContext secAuthorization ins}\]
\[\text{INS ACTIONS_IN}\]
\[\text{(exec}\]
\[\text{[SOME (SLc (OMNI ssmActionsInComplete))];}\n\[\text{SOME (SLc (PL withdraw))]}\]
\[\text{(Out Actions_IN}\]
\[\text{(exec}\]
\[\text{[SOME (SLc (OMNI ssmActionsInComplete))];}\n\[\text{SOME (SLc (PL withdraw)))); : outs}) \leftrightarrow \]
\[\text{authenticationTest inputOK}\]
\[\text{[Name Omni says}\]
\[\text{prop (SOME (SLc (OMNI ssmActionsInComplete))));}\n\[\text{Name PlatoonLeader says}\]
\[\text{prop (SOME (SLc (PL withdraw))));]}\]
\[\text{CFGInterpret} \ (M, Oi, Os)\]
\[\text{(CFG inputOK secContext secAuthorization}\]
\[\text{([Name Omni says}\]
\[\text{prop (SOME (SLc (OMNI ssmActionsInComplete))));}\n\[\text{Name PlatoonLeader says}\]
\[\text{prop (SOME (SLc (PL withdraw)))); : ins} \text{ ACTIONS_IN}\]
\[\text{outs}) \land \]
\[\text{(M, Oi, Os) satList}\]
\[\text{[prop (SOME (SLc (OMNI ssmActionsInComplete))));}\n\[\text{prop (SOME (SLc (PL withdraw))));]}\]

[PlatoonLeader_ACTIONS_IN_exec_lemma]
\[\forall M \ Oi \ Os.\]
\[\text{CFGInterpret} \ (M, Oi, Os)\]
\[\text{(CFG inputOK secContext secAuthorization}\]
\[\text{([Name Omni says}\]
\[\text{prop (SOME (SLc (OMNI ssmActionsInComplete))));}\n\[\text{Name PlatoonLeader says}\]
\[\text{prop (SOME (SLc (PL withdraw)))); : ins} \text{ ACTIONS_IN}\]
outs) \Rightarrow
(M, O_i, O_s) \text{ satList}
prop\text{CommandList}

[Name Omni says
prop (SOME (S\text{Lc (OMNI ssmActionsInComplete))});
Name PlatoonLeader says prop (SOME (S\text{Lc (PL withdraw))})]

\text{PlatoonLeader\_ACTIONS\_IN\_trap\_justified\_lemma}
\vdash omniCommand \neq ssmActionsInComplete \Rightarrow
(s = A\text{CTIONS\_IN}) \Rightarrow
\forall \text{NS } \text{Out } M \ O_i \ O_s.
\text{TR } (M, O_i, O_s)

(trap
(inputList
[Name Omni says
prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})])
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})]):ins) A\text{CTIONS\_IN}
outs)
(CFG inputOK secContext secAuthorization ins
\text{NS A\text{CTIONS\_IN}
(trap
(inputList
[Name Omni says
prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})])
(Out A\text{CTIONS\_IN
(trap
(inputList
[Name Omni says
prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})]):
outs)) \iff
authenticationTest inputOK
[Name Omni says prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})] \land
\text{CFGInterpret } (M, O_i, O_s)
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (S\text{Lc (OMNI omniCommand))});
Name PlatoonLeader says
prop (SOME (S\text{Lc (PL withdraw))})]:ins) A\text{CTIONS\_IN
outs) \land (M, O_i, O_s) \text{ sat prop NONE}
Theorems

SSMCONDUCTORP THEORY

[PlatoonLeader_ACTIONS_IN_trap_justified_thm]
\[ \forall omniCommand \neq \text{ssmActionsInComplete} \Rightarrow \]
\[ (s = \text{ACTIONS_IN}) \Rightarrow \]
\[ \forall NS \ Out\ M\ Oi\ Os. \]
\[ \text{TR} \ (M, Oi, Os) \]
\[ (\text{trap} \]
\[ [\text{SOME (SLc (OMNI omniCommand))}; \]
\[ \text{SOME (SLc (PL withdraw)))] \]
\[ \text{CFG inputOK secContext secAuthorization} \]
\[ ([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL withdraw))))]}::\text{ins} \] \]
\[ \text{ACTIONS_IN} \]
\[ \text{outs}) \]
\[ \text{(CFG inputOK secContext secAuthorization ins} \]
\[ \text{(NS ACTIONS_IN} \]
\[ (\text{trap} \]
\[ [\text{SOME (SLc (OMNI omniCommand))}; \]
\[ \text{SOME (SLc (PL withdraw)))] \]
\[ \text{(Out ACTIONS_IN} \]
\[ (\text{trap} \]
\[ [\text{SOME (SLc (OMNI omniCommand))}; \]
\[ \text{SOME (SLc (PL withdraw))]}::\text{outs}) \] \]
\[ \text{authenticationTest inputOK} \]
\[ [\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL withdraw))))] \wedge \]
\[ \text{CFGInterpret} \ (M, Oi, Os) \]
\[ \text{(CFG inputOK secContext secAuthorization} \]
\[ ([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL withdraw))))]}::\text{ins} \] \]
\[ \text{ACTIONS_IN} \]
\[ \text{outs}) \wedge (M, Oi, Os) \text{ sat prop NONE} \]

[PlatoonLeader_ACTIONS_IN_trap_lemma]
\[ \forall omniCommand \neq \text{ssmActionsInComplete} \Rightarrow \]
\[ (s = \text{ACTIONS_IN}) \Rightarrow \]
\[ \forall M\ Oi\ Os. \]
\[ \text{CFGInterpret} \ (M, Oi, Os) \]
\[ \text{(CFG inputOK secContext secAuthorization} \]
\[ ([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand)))}; \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL withdraw))))]}::\text{ins} \] \]
\[ \text{ACTIONS_IN} \]
\[ \text{outs}) \Rightarrow \]
\[ (M, Oi, Os) \text{ sat prop NONE} \]

[PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm]
\[ \forall NS \ Out\ M\ Oi\ Os. \]
\[ \text{TR} \ (M, Oi, Os) \ (\text{exec [SOME (SLc (PL secure))])} \]
(CFG inputOK secContext secAuthorization
  ([Name PlatoonLeader says
    prop (SOME (SLc (PL secure))):ins) CONDUCT_ORP
   outs]
  (CFG inputOK secContext secAuthorization
   ins
   (NS CONDUCT_ORP (exec [SOME (SLc (PL secure))]))
   (Out CONDUCT_ORP (exec [SOME (SLc (PL secure))]):ins) outs))
  authenticationTest inputOK
  ([Name PlatoonLeader says prop (SOME (SLc (PL secure))))] ∧
  CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL secure))):ins) CONDUCT_ORP
     outs])
   (M, Oi, Os) satList [prop (SOME (SLc (PL secure)))]
  )

[PlatoonLeader_CONDUCT_ORP_exec_secure_lemma]
\[ \forall M Oi Os.\]

CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL secure))):ins) CONDUCT_ORP
     outs])
  (M, Oi, Os) satList
  propCommandList
  ([Name PlatoonLeader says prop (SOME (SLc (PL secure)))]

[PlatoonSergeant_SECURE_exec_justified_lemma]
\[ \forall NS Out M Oi Os.\]

TR (M, Oi, Os)
  (exec
  ([inputList
    [Name Omni says
      prop (SOME (SLc (OMNI ssmSecureComplete)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG actionsIn)))]])
  (CFG inputOK secContext secAuthorization
    ([Name Omni says
      prop (SOME (SLc (OMNI ssmSecureComplete)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG actionsIn))):ins) SECURE
     outs]
  (CFG inputOK secContext secAuthorization
   (NS SECURE
    (exec
     ([inputList
      [Name Omni says
        prop (SOME (SLc (OMNI ssmSecureComplete))]);
    ])}

42
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))

(Out SECURE
(exec
(inputList
 [Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
 Name PlatoonSergeant says
 prop (SOME (SLc (PSG actionsIn)))])):: outs)) \iff
authenticationTest inputOK
 [Name Omni says
 prop (SOME (SLc (OMNI ssmSecureComplete)));
 Name PlatoonSergeant says
 prop (SOME (SLc (PSG actionsIn)))] \land
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
 ([Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
 Name PlatoonSergeant says
 prop (SOME (SLc (PSG actionsIn)))]::ins) SECURE
outs) \land
(M, Oi, Os) satList
propCommandList
 [Name Omni says
 prop (SOME (SLc (OMNI ssmSecureComplete)));
 Name PlatoonSergeant says
 prop (SOME (SLc (PSG actionsIn)))]

[PlatoonSergeant_SECURE_exec_justified_thm]
\vdash \forall NS Out M Oi Os.
TR (M, Oi, Os)
(exec
 [SOME (SLc (OMNI ssmSecureComplete));
 SOME (SLc (PSG actionsIn))]
(CFG inputOK secContext secAuthorization
 ([Name Omni says
  prop (SOME (SLc (OMNI ssmSecureComplete)));
 Name PlatoonSergeant says
 prop (SOME (SLc (PSG actionsIn)))]::ins) SECURE
outs)
(CFG inputOK secContext secAuthorization ins
 (NS SECURE
 (exec
  [SOME (SLc (OMNI ssmSecureComplete));
   SOME (SLc (PSG actionsIn))])
(Out SECURE
 (exec
  [SOME (SLc (OMNI ssmSecureComplete))];
CONDUCTORPYPE THEORY

SOME (SLc (PSG actionsIn)))::outs) \iff
authenticationTest inputOK
\begin{itemize}
  \item Name Omni says
    \begin{itemize}
      \item prop (SOME (SLc (OMNI ssmSecureComplete)));\end{itemize}
  \item Name PlatoonSergeant says
    \begin{itemize}
      \item prop (SOME (SLc (PSG actionsIn))));\end{itemize}
\end{itemize}
\end{itemize}
\end{itemize}

\[\text{PlatoonSergeant\_SECURE\_exec\_lemma}\]
\[
\forall M\ O_i\ O_s. \\
\text{CFGInterpret} (M, O_i, O_s) \\
\begin{itemize}
  \item prop (SOME (SLc (OMNI ssmSecureComplete)));\end{itemize}
\end{itemize}
\end{itemize}

9 ConductORPType Theory

Built: 11 June 2018
Parent Theories: indexedLists, patternMatches

9.1 Datatypes

\begin{itemize}
  \item omniCommand = ssmSecureComplete | ssmActionsInComplete
  \item plCommand = secure | withdraw | complete | plIncomplete
  \item psgCommand = actionsIn | psgIncomplete
\end{itemize}
Theorems

\[\text{slCommand} = \]
\begin{align*}
\text{PL ConductORP} & \quad | \quad \text{PSG ConductORP} \\
\text{OMNI omniCommand} & \\
\text{slOutput} = \text{ConductORP} & \quad | \quad \text{Secure} & \quad | \quad \text{ActionsIn} & \quad | \quad \text{Withdraw} & \quad | \quad \text{Complete} \\
\text{unAuthenticated} & \quad | \quad \text{unAuthorized} \\
\text{slState} = \text{CONDUCT\_ORP} & \quad | \quad \text{SECURE} & \quad | \quad \text{ACTIONS\_IN} & \quad | \quad \text{WITHDRAW} \\
\text{COMPLETE} & \\
\text{stateRole} = \text{PlatoonLeader} & \quad | \quad \text{PlatoonSergeant} & \quad | \quad \text{Omni}
\end{align*}

### 9.2 Theorems

#### [omniCommand_distinct_clauses]
\[\vdash \text{ssmSecureComplete} \neq \text{ssmActionsInComplete} \land \]
\[\text{ssmSecureComplete} \neq \text{ssmWithdrawComplete} \land \]
\[\text{ssmActionsInComplete} \neq \text{ssmWithdrawComplete} \land \]
\[\text{ssmActionsInComplete} \neq \text{invalidOmniCommand} \land \]
\[\text{ssmWithdrawComplete} \neq \text{invalidOmniCommand} \]

#### [plCommand_distinct_clauses]
\[\vdash \text{secure} \neq \text{withdraw} \land \text{secure} \neq \text{complete} \land \]
\[\text{secure} \neq \text{plIncomplete} \land \text{withdraw} \neq \text{complete} \land \]
\[\text{withdraw} \neq \text{plIncomplete} \land \text{complete} \neq \text{plIncomplete} \]

#### [psgCommand_distinct_clauses]
\[\vdash \text{actionsIn} \neq \text{psgIncomplete} \]

#### [slCommand_distinct_clauses]
\[\vdash \forall a' a. \text{PL} a \neq \text{PSG} a' \land \]
\[\forall a' a. \text{PL} a \neq \text{OMNI} a' \land \]
\[\forall a' a. \text{PSG} a \neq \text{OMNI} a' \]

#### [slCommand_one_one]
\[\vdash \forall a a'. (\text{PL} a = \text{PL} a') \iff (a = a') \land \]
\[\forall a a'. (\text{PSG} a = \text{PSG} a') \iff (a = a') \land \]
\[\forall a a'. (\text{OMNI} a = \text{OMNI} a') \iff (a = a') \]

#### [slOutput_distinct_clauses]
\[\vdash \text{ConductORP} \neq \text{Secure} \land \text{ConductORP} \neq \text{ActionsIn} \land \]
\[\text{ConductORP} \neq \text{Withdraw} \land \text{ConductORP} \neq \text{Complete} \land \]
\[\text{ConductORP} \neq \text{unAuthenticated} \land \text{ConductORP} \neq \text{unAuthorized} \land \]
\[\text{Secure} \neq \text{ActionsIn} \land \text{Secure} \neq \text{Withdraw} \land \text{Secure} \neq \text{Complete} \land \]
\[\text{Secure} \neq \text{unAuthenticated} \land \text{Secure} \neq \text{unAuthorized} \land \]
\[\text{ActionsIn} \neq \text{Withdraw} \land \text{ActionsIn} \neq \text{Complete} \land \]
\[\text{ActionsIn} \neq \text{unAuthenticated} \land \text{ActionsIn} \neq \text{unAuthorized} \land \]
\[\text{Withdraw} \neq \text{Complete} \land \text{Withdraw} \neq \text{unAuthenticated} \land \]
\[\text{Withdraw} \neq \text{unAuthorized} \land \text{Complete} \neq \text{unAuthenticated} \land \]
\[\text{Complete} \neq \text{unAuthorized} \land \text{unAuthenticated} \neq \text{unAuthorized} \]

45
CONDUCTORPDEF THEORY

[slRole_distinct_clauses]
\[ \vdash \text{PlatoonLeader} \neq \text{PlatoonSergeant} \land \text{PlatoonLeader} \neq \text{Omni} \land \text{PlatoonSergeant} \neq \text{Omni} \]

[slState_distinct_clauses]
\[ \vdash \text{CONDUCT\_ORP} \neq \text{SECURE} \land \text{CONDUCT\_ORP} \neq \text{ACTIONS\_IN} \land \text{CONDUCT\_ORP} \neq \text{WITHDRAW} \land \text{CONDUCT\_ORP} \neq \text{COMPLETE} \land \text{SECURE} \neq \text{ACTIONS\_IN} \land \text{SECURE} \neq \text{WITHDRAW} \land \text{SECURE} \neq \text{COMPLETE} \land \text{ACTIONS\_IN} \neq \text{WITHDRAW} \land \text{ACTIONS\_IN} \neq \text{COMPLETE} \land \text{WITHDRAW} \neq \text{COMPLETE} \]

10 ConductORPDef Theory

**Built:** 11 June 2018

**Parent Theories:** ConductORPType, ssm, OMNIType

10.1 Definitions

[secAuthorization_def]
\[ \vdash \forall \text{xs}. \text{secAuthorization} \text{xs} = \text{secHelper} (\text{getOmniCommand} \text{xs}) \]

[secContext_def]
\[ \vdash (\forall \text{xs}. \text{secContext} \text{CONDUCT\_ORP} \text{xs} = \text{[Name PlatoonLeader controls \prop (SOME (SLc (PL secure)))]} \land \text{(\forall \text{xs}. \text{secContext} \text{SECURE} \text{xs} = \text{if \text{getOmniCommand} \text{xs} = \text{ssmSecureComplete} then \text{[prop (SOME (SLc (OMNI ssmSecureComplete))) impf \text{Name PlatoonSergeant controls \prop (SOME (SLc (PSG actionsIn))]} \text{else \[prop NONE\]}} \land \text{(\forall \text{xs}. \text{secContext} \text{ACTIONS\_IN} \text{xs} = \text{if \text{getOmniCommand} \text{xs} = \text{ssmActionsInComplete} then \text{[prop (SOME (SLc (OMNI ssmActionsInComplete))) impf \text{Name PlatoonLeader controls \prop (SOME (SLc (PL withdraw))]} \text{else \[prop NONE\]}} \land \text{(\forall \text{xs}. \text{secContext} \text{WITHDRAW} \text{xs} = \text{if \text{getOmniCommand} \text{xs} = \text{ssmWithdrawComplete} then \text{[prop (SOME (SLc (OMNI ssmWithdrawComplete))) impf \text{Name PlatoonLeader controls \prop (SOME (SLc (PL complete))]} \text{else \[prop NONE\]}}\]

46
10.2 Theorems

[secHelper_def]
\[ \forall \text{cmd}. \]
\[ \text{secHelper \ cmd} = \]
\[ \text{[Name Omni controls prop (SOME (SLc (OMNI \ cmd)))]} \]

[getOmniCommand_def]
\[ \vdash (\text{getOmniCommand} \ [\text{\}] = \text{invalidOmniCommand}) \land \]
\[ (\forall \text{xs \ cmd}. \)
\[ \text{getOmniCommand} \]
\[ \quad (\text{Name Omni says prop (SOME (SLc (OMNI \ cmd)))::xs}) = \]
\[ \text{cmd}) \land \]
\[ (\forall \text{xs, getOmniCommand} (\text{TT::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs, getOmniCommand} (\text{FF::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v2, getOmniCommand} (\text{prop \ v2::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v3, getOmniCommand} (\text{notf \ v3::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v5, getOmniCommand} (\text{v5 andf \ v5::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v7 \ v6, getOmniCommand} (\text{v6 orf \ v7::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v8, getOmniCommand} (\text{v8 impf \ v9::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v11 \ v10, getOmniCommand} (\text{v10 eqf \ v11::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v12, getOmniCommand} (\text{v12 says TT::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v12, getOmniCommand} (\text{v12 says FF::xs}) = \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v134, getOmniCommand} (\text{Name \ v134 says prop NONE::xs}) = \]
\[ \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v144, getOmniCommand} (\text{Name \ PlatoonLeader says prop (SOME \ v144)\::\::xs}) = \]
\[ \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v144, getOmniCommand} (\text{Name \ PlatoonSergeant says prop (SOME \ v144)\::\::xs}) = \]
\[ \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v146, getOmniCommand} (\text{Name \ Omni says prop (SOME (ESCc \ v146))\::\::xs}) = \]
\[ \text{getOmniCommand} \text{ xs}) \land \]
\[ (\forall \text{xs \ v150, getOmniCommand} (\text{Name \ Omni says prop (SOME (SLc \ (P\text{L v150}))\::\::xs}) = \]
\[ \text{getOmniCommand} \text{ xs}) \land \]
CONDUCTORPDEF THEORY

Theorems

(∀ xs v151.
  getOmniCommand
    (Name Omni says prop (SOME (SLc (PSG v151))):xs) =
    getOmniCommand xs) ∧
(∀ xs v08 v136 v135.
  getOmniCommand (v135 meet v136 says prop v08::xs) =
  getOmniCommand xs) ∧
(∀ xs v08 v138 v137.
  getOmniCommand (v137 quoting v138 says prop v08::xs) =
  getOmniCommand xs) ∧
(∀ xs v09 v12.
  getOmniCommand (v12 says notf v09::xs) =
  getOmniCommand xs) ∧
(∀ xs v71 v70 v12.
  getOmniCommand (v12 says (v70 andf v71)::xs) =
  getOmniCommand xs) ∧
(∀ xs v73 v72 v12.
  getOmniCommand (v12 says (v72 orf v73)::xs) =
  getOmniCommand xs) ∧
(∀ xs v75 v74 v12.
  getOmniCommand (v12 says (v74 impf v75)::xs) =
  getOmniCommand xs) ∧
(∀ xs v77 v76 v12.
  getOmniCommand (v12 says (v76 eqf v77)::xs) =
  getOmniCommand xs) ∧
(∀ xs v79 v78 v12.
  getOmniCommand (v12 says v78 says v79::xs) =
  getOmniCommand xs) ∧
(∀ xs v80 v81 v12.
  getOmniCommand (v12 says v80 speaks_for v81::xs) =
  getOmniCommand xs) ∧
(∀ xs v82 v83 v12.
  getOmniCommand (v12 says v82 controls v83::xs) =
  getOmniCommand xs) ∧
(∀ xs v84 v85 v86 v12.
  getOmniCommand (v12 says reps v84 v85 v86::xs) =
  getOmniCommand xs) ∧
(∀ xs v87 v88 v12.
  getOmniCommand (v12 says v87 domi v88::xs) =
  getOmniCommand xs) ∧
(∀ xs v89 v90 v12.
  getOmniCommand (v12 says v89 eqi v90::xs) =
  getOmniCommand xs) ∧
(∀ xs v91 v92 v12.
  getOmniCommand (v12 says v91 doms v92::xs) =
  getOmniCommand xs) ∧
(∀ xs v93 v94 v12.
  getOmniCommand (v12 says v93 eqs v94::xs) =
  getOmniCommand xs) ∧
(∀ z s v_{96} v_{95} v_{12}.  
getOmniCommand (v_{12} says v_{95} eqn v_{96}::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{98} v_{97} v_{12}.  
getOmniCommand (v_{12} says v_{97} lte v_{98}::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{99} v_{12} v100.  
getOmniCommand (v_{12} says v_{99} lt v100::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{15} v_{14}.  
getOmniCommand (v_{14} speaks_for v_{15}::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{17} v_{16}.  
getOmniCommand (v_{16} controls v_{17}::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{20} v_{19} v_{18}.  
getOmniCommand (reps v_{18} v_{19} v_{20}::zs) =  
getOmniCommand zs) \land  
(∀ z s v_{22} v_{21}.  
getOmniCommand (v_{21} domi v_{22}::zs) = getOmniCommand zs) \land  
(∀ z s v_{24} v_{23}.  
getOmniCommand (v_{23} eqi v_{24}::zs) = getOmniCommand zs) \land  
(∀ z s v_{26} v_{25}.  
getOmniCommand (v_{25} doms v_{26}::zs) = getOmniCommand zs) \land  
(∀ z s v_{28} v_{27}.  
getOmniCommand (v_{27} eqs v_{28}::zs) = getOmniCommand zs) \land  
(∀ z s v_{30} v_{29}.  
getOmniCommand (v_{29} eqn v_{30}::zs) = getOmniCommand zs) \land  
(∀ z s v_{32} v_{31}.  
getOmniCommand (v_{31} lte v_{32}::zs) = getOmniCommand zs) \land  
∀ z s v_{34} v_{33}.  
getOmniCommand (v_{33} lt v_{34}::zs) = getOmniCommand zs

[getOmniCommand_ind]  
\vdash \forall P,  
P [] \land  
(\forall cmd zs,  
P (Name Omni says prop (SOME (SLc (OMNI cmd)))::zs)) \land  
(\forall zs. P zs \Rightarrow P (TT::zs)) \land  (\forall zs. P zs \Rightarrow P (FF::zs)) \land  
(\forall v_{2} zs. P zs \Rightarrow P (prop v_{2}::zs)) \land  
(\forall v_{3} zs. P zs \Rightarrow P (notf v_{3}::zs)) \land  
(\forall v_{4} v_{5} zs. P zs \Rightarrow P (v_{4} andf v_{5}::zs)) \land  
(\forall v_{6} v_{7} zs. P zs \Rightarrow P (v_{6} orf v_{7}::zs)) \land  
(\forall v_{8} v_{9} zs. P zs \Rightarrow P (v_{8} impf v_{9}::zs)) \land  
(\forall v_{10} v_{11} zs. P zs \Rightarrow P (v_{10} eqf v_{11}::zs)) \land  
(\forall v_{12} zs. P zs \Rightarrow P (v_{12} says TT::zs)) \land  
(\forall v_{12} zs. P zs \Rightarrow P (v_{12} says FF::zs)) \land  
(\forall v_{134} zs. P zs \Rightarrow P (Name v_{134} says prop NONE::zs)) \land  
(\forall v_{144} zs.
\[ P \text{ xs } \Rightarrow \\
P \text{ (Name PlatoonLeader says prop (SOME v144::xs)) } \wedge \\
(\forall v144 \text{ xs}. \\
P \text{ xs } \Rightarrow \\
P \text{ (Name PlatoonSergeant says prop (SOME v144::xs)) } \wedge \\
(\forall v146 \text{ xs}. \\
P \text{ xs } \Rightarrow P \text{ (Name Omni says prop (SOME (ESCc v146)::xs)) } \wedge \\
(\forall v150 \text{ xs}. \\
P \text{ xs } \Rightarrow \\
P \text{ (Name Omni says prop (SOME (SLc (PL v150))::xs)) } \wedge \\
(\forall v151 \text{ xs}. \\
P \text{ xs } \Rightarrow \\
P \text{ (Name Omni says prop (SOME (SLc (PSG v151))::xs)) } \wedge \\
(\forall v135 \text{ v136 v98 xs}. \\
P \text{ xs } \Rightarrow P \text{ (v135 meet v136 says prop v98::xs)) } \wedge \\
(\forall v137 \text{ v138 v98 xs}. \\
P \text{ xs } \Rightarrow P \text{ (v137 quoting v138 says prop v98::xs)) } \wedge \\
(\forall v12 \text{ v99 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says notf v99::xs)) } \wedge \\
(\forall v12 \text{ v70 v71 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says (v70 andf v71)::xs)) } \wedge \\
(\forall v12 \text{ v72 v73 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says (v72 orf v73)::xs)) } \wedge \\
(\forall v12 \text{ v74 v75 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says (v74 impf v75)::xs)) } \wedge \\
(\forall v12 \text{ v76 v77 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says (v76 eqf v77)::xs)) } \wedge \\
(\forall v12 \text{ v78 v79 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v78 says v79::xs)) } \wedge \\
(\forall v12 \text{ v80 v81 xs}. \\
P \text{ xs } \Rightarrow P \text{ (v12 says v80 speaks_for v81::xs)) } \wedge \\
(\forall v12 \text{ v82 v83 xs}. \\
P \text{ xs } \Rightarrow P \text{ (v12 says v82 controls v83::xs)) } \wedge \\
(\forall v12 \text{ v84 v85 v86 xs}. \\
P \text{ xs } \Rightarrow P \text{ (v12 says reps v84 v85 v86::xs)) } \wedge \\
(\forall v12 \text{ v87 v88 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v87 domi v88::xs)) } \wedge \\
(\forall v12 \text{ v89 v90 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v89 eqi v90::xs)) } \wedge \\
(\forall v12 \text{ v91 v92 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v91 doms v92::xs)) } \wedge \\
(\forall v12 \text{ v93 v94 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v93 eqs v94::xs)) } \wedge \\
(\forall v12 \text{ v95 v96 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v95 eqn v96::xs)) } \wedge \\
(\forall v12 \text{ v97 v98 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v97 lte v98::xs)) } \wedge \\
(\forall v12 \text{ v99 v100 xs}. P \text{ xs } \Rightarrow P \text{ (v12 says v99 lt v100::xs)) } \wedge \\
(\forall v14 \text{ v15 xs}. P \text{ xs } \Rightarrow P \text{ (v14 speaks_for v15::xs)) } \wedge \\
(\forall v14 \text{ v16 v17 xs}. P \text{ xs } \Rightarrow P \text{ (v16 controls v17::xs)) } \wedge \\
(\forall v14 \text{ v18 v19 v20 xs}. P \text{ xs } \Rightarrow P \text{ (reps v18 v19 v20::xs)) } \wedge \\
(\forall v21 \text{ v22 xs}. P \text{ xs } \Rightarrow P \text{ (v21 domi v22::xs)) } \wedge \\
(\forall v21 \text{ v24 xs}. P \text{ xs } \Rightarrow P \text{ (v23 eqi v24::xs)) } \wedge \\
(\forall v25 \text{ v26 xs}. P \text{ xs } \Rightarrow P \text{ (v25 doms v26::xs)) } \wedge \\
(\forall v27 \text{ v28 xs}. P \text{ xs } \Rightarrow P \text{ (v27 eqs v28::xs)) } \wedge \\
(\forall v29 \text{ v30 xs}. P \text{ xs } \Rightarrow P \text{ (v29 eqn v30::xs)) } \wedge \\
(\forall v31 \text{ v32 xs}. P \text{ xs } \Rightarrow P \text{ (v31 lte v32::xs)) } \wedge \\
(\forall v33 \text{ v34 xs}. P \text{ xs } \Rightarrow P \text{ (v33 lt v34::xs)) } \Rightarrow \\
\forall v. P \text{ v} \]
SSMCONDUCTPB THEORY

\[\begin{align*}
\vdash (\text{getPlCom} \; \square = \text{plIncomplete}) & \land \\
(\forall \, xs \; \text{cmd} \cdot \text{getPlCom} (\text{SOME} (\text{SLc (PL cmd)}):xs) = \text{cmd}) & \land \\
(\forall \, xs \cdot \text{getPlCom} (\text{NONE}:xs) = \text{getPlCom} \; xs) & \land \\
(\forall \, v_4 \cdot \text{getPlCom} (\text{SOME} (\text{ESCc} \; v_4):xs) = \text{getPlCom} \; xs) & \land \\
(\forall \, v_9 \cdot \text{getPlCom} (\text{SOME} (\text{SLc (PSG v_9)}):xs) = \text{getPlCom} \; xs) & \land \\
(\forall \, v_{10} \cdot \text{getPlCom} (\text{SOME} (\text{SLc (OMNI v_{10}})):xs) = \text{getPlCom} \; xs)
\end{align*}\]

\[
\text{[getPlCom\_ind]}
\]

\[\begin{align*}
\vdash \forall \, P, \\
P \; \square & \land (\forall \, \text{cmd} \; xs \cdot P \; (\text{SOME} (\text{SLc (PL cmd)}):xs)) \land \\
(\forall \, \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{NONE}:xs)) & \land \\
(\forall \, v_4 \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{ESCc} \; v_4):xs)) & \land \\
(\forall \, v_9 \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{SLc (PSG v_9)}):xs)) & \land \\
(\forall \, v_{10} \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{SLc (OMNI v_{10}})):xs)) & \Rightarrow \\
\forall \; v. \; P \; v
\end{align*}\]

\[
\text{[getPsgCom\_def]}
\]

\[\begin{align*}
\vdash (\text{getPsgCom} \; \square = \text{psgIncomplete}) & \land \\
(\forall \, \text{cmd} \cdot \text{getPsgCom} (\text{SOME} (\text{SLc (PSG cmd)}):xs) = \text{cmd}) & \land \\
(\forall \, \text{xs}. \; \text{getPsgCom} (\text{NONE}:xs) = \text{getPsgCom} \; xs) & \land \\
(\forall \, v_4 \cdot \text{getPsgCom} (\text{SOME} (\text{ESCc} \; v_4):xs) = \text{getPsgCom} \; xs) & \land \\
(\forall \, v_9 \cdot \text{getPsgCom} (\text{SOME} (\text{SLc (PL v_9)}):xs) = \text{getPsgCom} \; xs) & \land \\
(\forall \, v_{10} \cdot \text{getPsgCom} (\text{SOME} (\text{SLc (OMNI v_{10}})):xs) = \text{getPsgCom} \; xs)
\end{align*}\]

\[
\text{[getPsgCom\_ind]}
\]

\[\begin{align*}
\vdash \forall \, P, \\
P \; \square & \land (\forall \, \text{cmd} \; xs \cdot P \; (\text{SOME} (\text{SLc (PSG cmd)}):xs)) \land \\
(\forall \, \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{NONE}:xs)) & \land \\
(\forall \, v_4 \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{ESCc} \; v_4):xs)) & \land \\
(\forall \, v_8 \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{SLc (PL v_8)}):xs)) & \land \\
(\forall \, v_{10} \; \text{xs}. \; P \; \text{xs} \Rightarrow P \; (\text{SOME} (\text{SLc (OMNI v_{10}})):xs)) & \Rightarrow \\
\forall \; v. \; P \; v
\end{align*}\]

11 ssmConductPB Theory

Built: 10 June 2018
Parent Theories: ConductPBType, ssm11, OMNIType

11.1 Definitions

\[
\text{[secContextConductPB\_def]}
\]

\[\begin{align*}
\vdash \forall \, \text{plcmd} \; \text{psgcmd} \; \text{incomplete}. \\
\text{secContextConductPB} \; \text{plcmd} \; \text{psgcmd} \; \text{incomplete} & = \\
[\text{Name PlatoonLeader controls prop (SOME (SLc (PL plcmd))}); \\
\text{Name PlatoonSergeant controls} \\
\text{prop (SOME (SLc (PSG psgcmd))}); \\
\text{Name PlatoonLeader says}
\end{align*}\]
prop (SOME (SLc (PSG psgcmd))) impf prop NONE;
Name PlatoonSergeant says
prop (SOME (SLc (PL plcmd))) impf prop NONE]

\[ [\text{ssmConductPBStateInterp_def} ] \]
\[ \vdash \forall \text{slState}. \ \text{ssmConductPBStateInterp} \ \text{slState} = \text{TT} \]

11.2 Theorems

\[ [\text{authTestConductPB_cmd_reject_lemma} ] \]
\[ \vdash \forall \text{cmd}. \ \neg \text{authTestConductPB} \ (\text{prop} \ (\text{SOME} \ \text{cmd})) \]

\[ [\text{authTestConductPB_def} ] \]
\[ \vdash (\text{authTestConductPB} \ (\text{Name PlatoonLeader says} \ \text{prop} \ \text{cmd}) \iff T) \land \\
(\text{authTestConductPB} \ (\text{Name PlatoonSergeant says} \ \text{prop} \ \text{cmd}) \iff T) \land (\text{authTestConductPB} \ \text{TT} \iff F) \land \\
(\text{authTestConductPB} \ \text{FF} \iff F) \land \\
(\text{authTestConductPB} \ (\text{prop} \ v) \iff F) \land \\
(\text{authTestConductPB} \ (\text{notf} \ v) \iff F) \land \\
(\text{authTestConductPB} \ (v_2 \ \text{andf} \ v_3) \iff F) \land \\
(\text{authTestConductPB} \ (v_4 \ \text{orf} \ v_5) \iff F) \land \\
(\text{authTestConductPB} \ (v_6 \ \text{impf} \ v_7) \iff F) \land \\
(\text{authTestConductPB} \ (v_8 \ \text{eqf} \ v_9) \iff F) \land \\
(\text{authTestConductPB} \ (v_{10} \ \text{says} \ \text{TT}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{10} \ \text{says} \ \text{FF}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{1133} \ \text{meet} \ v_{1134} \ \text{says} \ \text{prop} \ v_{1166}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{1135} \ \text{quoting} \ v_{1136} \ \text{says} \ \text{prop} \ v_{1166}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ \text{notf} \ v_{1107}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ (v_{1168} \ \text{andf} \ v_{1169})) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ (v_{1170} \ \text{orf} \ v_{1171})) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ (v_{1172} \ \text{impf} \ v_{1173})) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ (v_{1174} \ \text{eqf} \ v_{1175})) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ (v_{1176} \ \text{says} \ v_{1177}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1178} \ \text{speaks_for} \ v_{1179}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1180} \ \text{controls} \ v_{1181}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1180} \ \text{controls} \ v_{1181}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1180} \ \text{controls} \ v_{1181}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1187} \ \text{eqi} \ v_{1188}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1189} \ \text{doms} \ v_{1190}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1191} \ \text{eqs} \ v_{1192}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1193} \ \text{eqn} \ v_{1194}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1195} \ \text{lte} \ v_{1196}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{says} \ v_{1197} \ \text{lt} \ v_{1198}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{112} \ \text{speaks_for} \ v_{1113}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{114} \ \text{controls} \ v_{1115}) \iff F) \land \\
(\text{authTestConductPB} \ (\text{reps} \ v_{1116} \ v_{1117} \ v_{1118}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{110} \ \text{doms} \ v_{1120}) \iff F) \land \\
(\text{authTestConductPB} \ (v_{1121} \ \text{eqi} \ v_{1122}) \iff F) \land 
(authTestConductPB (v_{21} \text{ doms } v_{24}) \iff F) \land
(authTestConductPB (v_{20} \text{ eqs } v_{26}) \iff F) \land
(authTestConductPB (v_{27} \text{ eqn } v_{28}) \iff F) \land
(authTestConductPB (v_{20} \text{ lte } v_{30}) \iff F) \land
(authTestConductPB (v_{21} \text{ lt } v_{22}) \iff F)

[authTestConductPB\_ind]
\vdash \forall P.
(\forall \text{cmd}. \ P (\text{Name PlatoonLeader says prop cmd})) \land
(\forall \text{cmd}. \ P (\text{Name PlatoonSergeant says prop cmd})) \land P \text{ TT} \land
P \text{ FF} \land (\forall v. \ P (\text{prop } v)) \land (\forall v_1. \ P (\text{notf } v_1)) \land
(\forall v_2. \ P (v_2 \text{ andf } v_3)) \land (\forall v_4. \ P (v_4 \text{ orf } v_5)) \land
(\forall v_6. \ P (v_6 \text{ impf } v_7)) \land (\forall v_8. \ P (v_8 \text{ eqf } v_9)) \land
(\forall v_{10}. \ P (v_{10} \text{ says } T)) \land (\forall v_{10}. \ P (v_{10} \text{ says } F)) \land
(\forall v_{133}. \ v_{134}. \ v_{96}. \ P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{96})) \land
(\forall v_{135}. \ v_{136}. \ v_{96}. \ P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{96})) \land
(\forall v_{96}. \ P (v_{10} \text{ says notf } v_{97})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{84} \text{ andf } v_{89})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{70} \text{ orf } v_{71})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{72} \text{ impf } v_{73})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{74} \text{ eqf } v_{75})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{70})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{82} \text{ reps } v_{83} \text{ v}_{84})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \land
(\forall v_{10}. \ P (v_{10} \text{ says } v_{97} \text{ lte } v_{98})) \land
(\forall v_{12}. \ P (v_{12} \text{ speaks_for } v_{13})) \land
(\forall v_{14}. \ P (v_{14} \text{ controls } v_{15})) \land
(\forall v_{16}. \ P (\text{reps } v_{16} \text{ v}_{17} \text{ v}_{18})) \land
(\forall v_{19}. \ P (v_{19} \text{ domi } v_{20})) \land
(\forall v_{21}. \ P (v_{21} \text{ eqi } v_{22})) \land
(\forall v_{23}. \ P (v_{23} \text{ doms } v_{24})) \land
(\forall v_{25}. \ P (v_{25} \text{ eqs } v_{26})) \land (\forall v_{27}. \ P (v_{27} \text{ eqn } v_{28})) \land
(\forall v_{29}. \ P (v_{29} \text{ lte } v_{30})) \land (\forall v_{31}. \ P (v_{31} \text{ lt } v_{32})) \supseteq
\forall v. \ P

[conductPBNS\_def]
\vdash \text{(conductPBNS CONDUCT_P}} \text{(exec (PL securePB)) = SECURE_P) \land
\text{(conductPBNS CONDUCT_P}} \text{(exec (PL p1IncompletePB)) = CONDUCT_P) \land
\text{(conductPBNS SECURE_P}} \text{(exec (PSG actionsInPB)) = ACTIONS_IN_P) \land
\text{(conductPBNS SECURE_P}} \text{(exec (PSG psgIncompletePB)) =}
SSMCONDUCTPB THEORY

Theorems

SECURE_PB) ∧
(conductPBNS ACTIONS_IN_PB (exec (PL withdrawPB)) =
WITHDRAW_PB) ∧
(conductPBNS ACTIONS_IN_PB (exec (PL plIncompletePB)) =
ACTIONS_IN_PB) ∧
(conductPBNS WITHDRAW_PB (exec (PL completePB)) =
COMPLETE_PB) ∧
(conductPBNS WITHDRAW_PB (exec (PL plIncompletePB)) =
WITHDRAW_PB) ∧ (conductPBNS s (trap (PL cmd')) = s) ∧
(conductPBNS s (trap (PSG cmd)) = s) ∧
(conductPBNS s (discard (PL cmd')) = s) ∧
(conductPBNS s (discard (PSG cmd)) = s)

[conductPBNS_ind]

| P CONDUCT_PB (exec (PL securePB)) ∧
P CONDUCT_PB (exec (PL plIncompletePB)) ∧
P SECURE_PB (exec (PSG actionsInPB)) ∧
P SECURE_PB (exec (PSG psgIncompletePB)) ∧
P ACTIONS_IN_PB (exec (PL withdrawPB)) ∧
P ACTIONS_IN_PB (exec (PL plIncompletePB)) ∧
P WITHDRAW_PB (exec (PL completePB)) ∧
P WITHDRAW_PB (exec (PL plIncompletePB)) ∧
(∀s cmd. P s (trap (PL cmd))) ∧
(∀s cmd. P s (trap (PSG cmd))) ∧
(∀s cmd. P s (discard (PL cmd))) ∧
(∀s cmd. P s (discard (PSG cmd))) ∧
P CONDUCT_PB (exec (PL withdrawPB)) ∧
P CONDUCT_PB (exec (PL completePB)) ∧
(∀v₁₁. P CONDUCT_PB (exec (PSG v₁₁))) ∧
(∀v₁₃. P SECURE_PB (exec (PL v₁₃))) ∧
P ACTIONS_IN_PB (exec (PL securePB)) ∧
P ACTIONS_IN_PB (exec (PL completePB)) ∧
(∀v₁₇. P ACTIONS_IN_PB (exec (PSG v₁₇))) ∧
P WITHDRAW_PB (exec (PL securePB)) ∧
P WITHDRAW_PB (exec (PL withdrawPB)) ∧
(∀v₂₀. P WITHDRAW_PB (exec (PSG v₂₀))) ∧
(∀v₂₁. P COMPLETE_PB (exec v₂₁)) ⇒
∀v₁. P v₁

[conductPBOut_def]

| (conductPBOut CONDUCT_PB (exec (PL securePB)) = ConductPB) ∧
(conductPBOut CONDUCT_PB (exec (PL plIncompletePB)) =
ConductPB) ∧
(conductPBOut SECURE_PB (exec (PSG actionsInPB)) =
SecurePB) ∧
(conductPBOut SECURE_PB (exec (PSG psgIncompletePB)) =
SecurePB) ∧
(conductPBOut ACTIONS_IN_PB (exec (PL withdrawPB)) =
\[
\text{ActionsInPB} \land \\
(\text{conductPBOut ACTIONS_IN_PB (exec (PL plIncompletePB))} = \\
\text{ActionsInPB}) \land \\
(\text{conductPBOut WITHDRAW_PB (exec (PL completePB))} = \\
\text{WithdrawPB}) \land \\
(\text{conductPBOut WITHDRAW_PB (exec (PL plIncompletePB))} = \\
\text{WithdrawPB}) \land \\
(\text{conductPBOut s (trap (PL cmd'))} = \text{unAuthorized}) \land \\
(\text{conductPBOut s (trap (PSG cmd')} = \text{unAuthorized}) \land \\
(\text{conductPBOut s (discard (PL cmd'))} = \text{unAuthenticated}) \land \\
(\text{conductPBOut s (discard (PSG cmd')} = \text{unAuthenticated})
\]

\[\text{conductPBOut_ind}\]
\[
\exists P. \\
P. P \text{CONDUCT_PB (exec (PL securePB))} \land \\
P. P \text{CONDUCT_PB (exec (PL plIncompletePB))} \land \\
P. P \text{SECURE_PB (exec (PSG actionsInPB))} \land \\
P. P \text{SECURE_PB (exec (PSG plIncompletePB))} \land \\
P. P \text{ACTIONS_IN_PB (exec (PL withdrawPB))} \land \\
P. P \text{ACTIONS_IN_PB (exec (PL plIncompletePB))} \land \\
P. P \text{WITHDRAW_PB (exec (PL completePB))} \land \\
P. P \text{WITHDRAW_PB (exec (PL plIncompletePB))} \land \\
(\forall s \text{ cmd}. P s (\text{trap (PL cmd')})) \land \\
(\forall s \text{ cmd}. P s (\text{trap (PSG cmd')})) \land \\
(\forall s \text{ cmd}. P s (\text{discard (PL cmd')})) \land \\
(\forall s \text{ cmd}. P s (\text{discard (PSG cmd')})) \land \\
P. P \text{CONDUCT_PB (exec (PL withdrawPB))} \land \\
P. P \text{CONDUCT_PB (exec (PL completePB))} \land \\
(\forall v_{11}. P \text{CONDUCT_PB (exec (PSG v_{11}))}) \land \\
(\forall v_{13}. P \text{SECURE_PB (exec (PL v_{13}))}) \land \\
P. P \text{ACTIONS_IN_PB (exec (PL securePB))} \land \\
P. P \text{ACTIONS_IN_PB (exec (PL completePB))} \land \\
(\forall v_{17}. P \text{ACTIONS_IN_PB (exec (PSG v_{17}))}) \land \\
P. P \text{WITHDRAW_PB (exec (PL securePB))} \land \\
P. P \text{WITHDRAW_PB (exec (PL withdrawPB))} \land \\
(\forall v_{20}. P \text{WITHDRAW_PB (exec (PSG v_{20}))}) \land \\
(\forall v_{21}. P \text{COMPLETE_PB (exec v_{21})}) \Rightarrow \\
\forall v v_{1}. P v v_{1}
\]

\[\text{PlatoonLeader_exec_plCommandPB_justified_thm}\]
\[
\exists NS. \text{Out M Oi Os}. \\
\text{TR (M, Oi, Os) (exec (SLc (PL plCommand)))} \\
(\text{CFG authTestConductPB ssmConductPBStateInterp} \\
(\text{secContextConductPB plCommand psgCommand incomplete}) \\
(\text{Name PlatoonLeader says} \\
prop (\text{SOME (SLc (PL plCommand))::ins} s outs) \\
(\text{CFG authTestConductPB ssmConductPBStateInterp} \\
(\text{secContextConductPB plCommand psgCommand incomplete}) \\
ins (NS s (exec (SLc (PL plCommand))))
\]

55
(\(\text{Out} \, s \, (\text{exec} \, (\text{SLc} \, (\text{PL} \, \text{plCommand}))))::\text{outs}) \iff \\
\text{authTestConductPB} \\
(\text{Name} \, \text{PlatoonLeader} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PL} \, \text{plCommand})))) \land \\
\text{CFGInterpret} \, (M, O_i, O_s) \\
(\text{CFG} \, \text{authTestConductPB} \, \text{ssmConductPBStateInterp} \\
(\text{secContextConductPB} \, \text{plCommand} \, \text{psgCommand} \, \text{incomplete}) \\
(\text{Name} \, \text{PlatoonLeader} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PL} \, \text{plCommand}))))::\text{ins} \, s \, \text{outs}) \land \\
(M, O_i, O_s) \, \text{sat} \, \text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PL} \, \text{plCommand}))))

[PlatoonLeader_plCommandPB_lemma]

\(\vdash \, \forall \, NS \, \text{Out} \, M \, O_i \, O_s. \\
\text{TR} \, (M, O_i, O_s) \, (\text{exec} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))) \\
(\text{CFG} \, \text{authTestConductPB} \, \text{ssmConductPBStateInterp} \\
(\text{secContextConductPB} \, \text{plCommand} \, \text{psgCommand} \, \text{incomplete}) \\
(\text{Name} \, \text{PlatoonSergeant} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))::\text{ins} \, s \, \text{outs} \\
(M, O_i, O_s) \, \text{sat} \, \text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PL} \, \text{plCommand})))))

[PlatoonSergeant_exec_psgCommandPB_justified_thm]

\(\vdash \, \forall \, NS \, \text{Out} \, M \, O_i \, O_s. \\
\text{TR} \, (M, O_i, O_s) \, (\text{exec} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))) \\
(\text{CFG} \, \text{authTestConductPB} \, \text{ssmConductPBStateInterp} \\
(\text{secContextConductPB} \, \text{plCommand} \, \text{psgCommand} \, \text{incomplete}) \\
(\text{Name} \, \text{PlatoonSergeant} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))::\text{ins} \, s \, \text{outs} \\
\text{ins} \, (\text{NS} \, s \, (\text{exec} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand})))) \\
(\text{Out} \, s \, (\text{exec} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))::\text{outs}) \iff \\
\text{authTestConductPB} \\
(\text{Name} \, \text{PlatoonSergeant} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand})))) \land \\
\text{CFGInterpret} \, (M, O_i, O_s) \\
(\text{CFG} \, \text{authTestConductPB} \, \text{ssmConductPBStateInterp} \\
(\text{secContextConductPB} \, \text{plCommand} \, \text{psgCommand} \, \text{incomplete}) \\
(\text{Name} \, \text{PlatoonSergeant} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))::\text{ins} \, s \, \text{outs}) \land \\
(M, O_i, O_s) \, \text{sat} \, \text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))

[PlatoonSergeant_psgCommandPB_lemma]

\(\vdash \, \forall \, \text{CFGInterpret} \, (M, O_i, O_s) \\
(\text{CFG} \, \text{authTestConductPB} \, \text{ssmConductPBStateInterp} \\
(\text{secContextConductPB} \, \text{plCommand} \, \text{psgCommand} \, \text{incomplete}) \\
(\text{Name} \, \text{PlatoonSergeant} \, \text{says} \\
\text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))::\text{ins} \, s \, \text{outs} \\
(M, O_i, O_s) \, \text{sat} \, \text{prop} \, (\text{SOME} \, (\text{SLc} \, (\text{PSG} \, \text{psgCommand}))))

56
12 ConductPBType Theory

**Built:** 10 June 2018

**Parent Theories:** indexedLists, patternMatches

### 12.1 Datatypes

\[ \text{plCommandPB} = \text{securePB} \mid \text{withdrawPB} \mid \text{completePB} \mid \text{plIncompletePB} \]

\[ \text{psgCommandPB} = \text{actionsInPB} \mid \text{psgIncompletePB} \]

\[ \text{slCommand} = \text{PL plCommandPB} \mid \text{PSG psgCommandPB} \]

\[ \text{slOutput} = \text{ConductPB} \mid \text{SecurePB} \mid \text{ActionsInPB} \mid \text{WithdrawPB} \mid \text{CompletePB} \mid \text{unAuthenticated} \mid \text{unAuthorized} \]

\[ \text{slState} = \text{CONDUCT_PB} \mid \text{SECURE_PB} \mid \text{ACTIONS_IN_PB} \mid \text{WITHDRAW_PB} \mid \text{COMPLETE_PB} \]

\[ \text{stateRole} = \text{PlatoonLeader} \mid \text{PlatoonSergeant} \]

### 12.2 Theorems

**[plCommandPB_distinct_clauses]**

\[ \vdash \text{securePB} \neq \text{withdrawPB} \land \text{securePB} \neq \text{completePB} \land \text{withdrawPB} \neq \text{plIncompletePB} \land \text{completePB} \neq \text{plIncompletePB} \]

**[psgCommandPB_distinct_clauses]**

\[ \vdash \text{actionsInPB} \neq \text{psgIncompletePB} \]

**[slCommand_distinct_clauses]**

\[ \forall a' a. \text{PL} a \neq \text{PSG} a' \]

**[slCommand_one_one]**

\[ \vdash (\forall a a'. (\text{PL} a = \text{PL} a') \iff (a = a')) \land (\forall a a'. (\text{PSG} a = \text{PSG} a') \iff (a = a')) \]

**[slOutput_distinct_clauses]**

\[ \vdash \text{ConductPB} \neq \text{SecurePB} \land \text{ConductPB} \neq \text{ActionsInPB} \land \text{ConductPB} \neq \text{WithdrawPB} \land \text{ConductPB} \neq \text{CompletePB} \land \text{SecurePB} \neq \text{ActionsInPB} \land \text{SecurePB} \neq \text{WithdrawPB} \land \text{SecurePB} \neq \text{CompletePB} \land \text{SecurePB} \neq \text{unAuthenticated} \land \text{SecurePB} \neq \text{unAuthorized} \land \text{ActionsInPB} \neq \text{CompletePB} \land \text{ActionsInPB} \neq \text{unAuthenticated} \land \text{ActionsInPB} \neq \text{unAuthorized} \land \text{WithdrawPB} \neq \text{unAuthenticated} \land \text{WithdrawPB} \neq \text{unAuthorized} \land \text{CompletePB} \neq \text{unAuthenticated} \land \text{CompletePB} \neq \text{unAuthorized} \land \text{unAuthenticated} \neq \text{unAuthorized} \]
13 ssmMoveToORP Theory

Built: 10 June 2018
Parent Theories: MoveToORPType, ssm11, OMNIType

13.1 Definitions

[secContextMoveToORP_def]
\( \forall \text{cmd}. \) secContextMoveToORP cmd = [Name PlatoonLeader controls prop (SOME (SLc cmd))]}

[ssmMoveToORPStateInterp_def]
\( \forall \text{state}. \) ssmMoveToORPStateInterp state = TT

13.2 Theorems

[authTestMoveToORP_cmd_reject_lemma]
\( \forall \text{cmd} . \) \( \neg \) authTestMoveToORP (prop (SOME cmd))

[authTestMoveToORP_def]
\( \forall \text{cmd} . \) (authTestMoveToORP (Name PlatoonLeader says prop cmd) \( \iff \) T) \( \land \)
(authTestMoveToORP (prop v) \( \iff \) F) \( \land \)
(authTestMoveToORP (notf v) \( \iff \) F) \( \land \)
(authTestMoveToORP (v2 andf v3) \( \iff \) F) \( \land \)
(authTestMoveToORP (v4 orf v5) \( \iff \) F) \( \land \)
(authTestMoveToORP (v6 impf v7) \( \iff \) F) \( \land \)
(authTestMoveToORP (v8 eqf v9) \( \iff \) F) \( \land \)
(authTestMoveToORP (v10 says TT) \( \iff \) F) \( \land \)
(authTestMoveToORP (v10 says FF) \( \iff \) F) \( \land \)
(authTestMoveToORP (v133 meet v134 says prop v66) \( \iff \) F) \( \land \)
(authTestMoveToORP (v135 quoting v136 says prop v66) \( \iff \) F) \( \land \)
(authTestMoveToORP (v10 says notf v67) \( \iff \) F) \( \land \)
(authTestMoveToORP (v10 says (v68 andf v69)) \( \iff \) F) \( \land \)
(authTestMoveToORP (v10 says (v70 orf v71)) \( \iff \) F) \( \land \)

58
(authTestMoveToORP \(v_{10} \text{ says } (v_{72} \text{ impf } v_{73}) \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}) \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{76} \text{ says } v_{77} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{80} \text{ controls } v_{81} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } \text{ reps } v_{82} v_{83} v_{84} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{85} \text{ domi } v_{86} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{87} \text{ eqi } v_{88} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{89} \text{ doms } v_{90} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{91} \text{ eqs } v_{92} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{93} \text{ eqn } v_{94} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{95} \text{ lte } v_{96} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{10} \text{ says } v_{97} \text{ lt } v_{98} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{12} \text{ speaks_for } v_{13} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{14} \text{ controls } v_{15} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(\text{ reps } v_{16} v_{17} v_{18} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{19} \text{ domi } v_{20} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{21} \text{ eqi } v_{22} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{23} \text{ doms } v_{24} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{25} \text{ eqs } v_{26} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{27} \text{ eqn } v_{28} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{29} \text{ lte } v_{30} \) \(\iff F\) \(\land\)  
(authTestMoveToORP \(v_{31} \text{ lt } v_{32} \) \(\iff F\) \(\land\)  

[authTestMoveToORP_ind]  
\(\vdash \forall cmd. P \) (Name PlatoonLeader says prop cmd)) \(\land\) P TT \(\land\)  
P FF \(\land\) (\(\forall v. P \) (prop v)) \(\land\) (\(\forall v_1. P \) (notf v_1)) \(\land\)  
(\(\forall v_2 v_3. P \) (v_2 andf v_3)) \(\land\) (\(\forall v_4 v_5. P \) (v_4 orf v_5)) \(\land\)  
(\(\forall v_6 v_7. P \) (v_6 impf v_7)) \(\land\) (\(\forall v_8 v_9. P \) (v_8 eqf v_9)) \(\land\)  
(\(\forall v_{10}. P \) (v_{10} says TT)) \(\land\) (\(\forall v_{10}. P \) (v_{10} says FF)) \(\land\)  
(\(\forall v_{133} v_{134} v_{96}. P \) (v_{133} meet v_{134} says prop v_{96})) \(\land\)  
(\(\forall v_{135} v_{136} v_{96}. P \) (v_{135} quoting v_{136} says prop v_{96})) \(\land\)  
(\(\forall v_{97}. P \) (v_{10} says notf v_{97})) \(\land\)  
(\(\forall v_{10} v_{88} v_{89}. P \) (v_{10} says (v_{88} andf v_{89}))) \(\land\)  
(\(\forall v_{10} v_{70} v_{71}. P \) (v_{10} says (v_{70} orf v_{71}))) \(\land\)  
(\(\forall v_{10} v_{72} v_{73}. P \) (v_{10} says (v_{72} impf v_{73}))) \(\land\)  
(\(\forall v_{10} v_{74} v_{75}. P \) (v_{10} says (v_{74} eqf v_{75}))) \(\land\)  
(\(\forall v_{10} v_{76} v_{77}. P \) (v_{10} says v_{76} says v_{77})) \(\land\)  
(\(\forall v_{10} v_{78} v_{79}. P \) (v_{10} says v_{78} speaks_for v_{79})) \(\land\)  
(\(\forall v_{10} v_{80} v_{81}. P \) (v_{10} says v_{80} controls v_{81}))) \(\land\)  
(\(\forall v_{10} v_{82} v_{83} v_{84}. P \) (v_{10} says reps v_{82} v_{83} v_{84})) \(\land\)  
(\(\forall v_{10} v_{85} v_{86}. P \) (v_{10} says v_{85} domi v_{86})) \(\land\)  
(\(\forall v_{10} v_{87} v_{88}. P \) (v_{10} says v_{87} eqi v_{88})) \(\land\)  
(\(\forall v_{10} v_{89} v_{90}. P \) (v_{10} says v_{89} doms v_{90})) \(\land\)  
(\(\forall v_{10} v_{91} v_{92}. P \) (v_{10} says v_{91} eqs v_{92})) \(\land\)  
(\(\forall v_{10} v_{93} v_{94}. P \) (v_{10} says v_{93} eqn v_{94})) \(\land\)  
(\(\forall v_{10} v_{95} v_{96}. P \) (v_{10} says v_{95} lte v_{96})) \(\land\)  
(\(\forall v_{10} v_{97} v_{98}. P \) (v_{10} says v_{97} lt v_{98})) \(\land\)  

59
(∀ v_{12} v_{13}. \ P\ (v_{12} \ is\ for\ v_{13})) \land
(∀ v_{14} v_{15}. \ P\ (v_{14} \ is\ controls\ v_{15})) \land
(∀ v_{16} v_{17} v_{18}. \ P\ (reps\ v_{16} v_{17} v_{18})) \land
(∀ v_{19} v_{20}. \ P\ (v_{19} \ domi\ v_{20})) \land
(∀ v_{21} v_{22}. \ P\ (v_{21} \ eqi\ v_{22})) \land
(∀ v_{23} v_{24}. \ P\ (v_{23} \ doms\ v_{24})) \land
(∀ v_{25} v_{26}. \ P\ (v_{25} \ eqs\ v_{26})) \land
(∀ v_{27} v_{28}. \ P\ (v_{27} \ eqn\ v_{28})) \land
(∀ v_{29} v_{30}. \ P\ (v_{29} \ lte\ v_{30})) \land
(∀ v_{31} v_{32}. \ P\ (v_{31} \ lt\ v_{32})) \Rightarrow
∀ v. \ P v

[moveToORPNS_def]
⊢ (moveToORPNS MOVE_TO_ORP (exec (SLc pltForm))) = PLT_FORM \land
(moveToORPNS MOVE_TO_ORP (exec (SLc incomplete))) = MOVE_TO_ORP \land
(moveToORPNS PLT_FORM (exec (SLc pltMove))) = PLT_MOVE \land
(moveToORPNS PLT_FORM (exec (SLc incomplete))) = PLT_FORM \land
(moveToORPNS PLT_MOVE (exec (SLc pltSecureHalt))) = PLT_SECURE_HALT \land
(moveToORPNS PLT_MOVE (exec (SLc incomplete))) = PLT_MOVE \land
(moveToORPNS PLT_SECURE_HALT (exec (SLc complete))) = COMPLETE \land
(moveToORPNS PLT_SECURE_HALT (exec (SLc incomplete))) = PLT_SECURE_HALT \land
(moveToORPNS s (trap (SLc cmd))) = s \land
(moveToORPNS s (discard (SLc cmd))) = s

[moveToORPNS_ind]
⊢ ∀ P.
P MOVE_TO_ORP (exec (SLc pltForm)) \land
P MOVE_TO_ORP (exec (SLc incomplete)) \land
P PLT_FORM (exec (SLc pltMove)) \land
P PLT_FORM (exec (SLc incomplete)) \land
P PLT_MOVE (exec (SLc pltSecureHalt)) \land
P PLT_MOVE (exec (SLc incomplete)) \land
P PLT_SECURE_HALT (exec (SLc complete)) \land
P PLT_SECURE_HALT (exec (SLc incomplete)) \land
(∀ s \ cmd. \ P s (trap (SLc cmd))) \land
(∀ s \ cmd. \ P s (discard (SLc cmd))) \land
(∀ s \ v_6. \ P s (discard (ESCc v_6))) \land
(∀ s \ v_9. \ P s (trap (ESCc v_9))) \land
(∀ v_{12}. \ P MOVE_TO_ORP (exec (ESCc v_{12}))) \land
P MOVE_TO_ORP (exec (SLc pltMove)) \land
P MOVE_TO_ORP (exec (SLc pltSecureHalt)) \land
P MOVE_TO_ORP (exec (SLc complete)) \land
(∀ v_{15}. \ P PLT_FORM (exec (ESCc v_{15}))) \land
P PLT_FORM (exec (SLc pltForm)) \land
P PLT_FORM (exec (SLc pltSecureHalt)) \land
P PLT_FORM (exec (SLc complete)) \land
(∀ v_{18}. \ P PLT_MOVE (exec (ESCc v_{18}))) \land
P PLT_MOVE (exec (SLc pltForm)) \land
Theorems

\[ P \text{ PLT\_MOVE (exec (SLc pltMove)) } \wedge \\
P \text{ PLT\_MOVE (exec (SLc complete)) } \wedge \\
(\forall v_{21}. P \text{ PLT\_SECURE\_HALT (exec (ESCc v_{21})) } \wedge \\
P \text{ PLT\_SECURE\_HALT (exec (SLc pltForm)) } \wedge \\
P \text{ PLT\_SECURE\_HALT (exec (SLc pltMove)) } \wedge \\
P \text{ PLT\_SECURE\_HALT (exec (SLc pltSecureHalt)) } \wedge \\
(\forall v_{23}. P \text{ COMPLETE (exec v_{23})) } \Rightarrow \\
\forall v. v_{1} P v v_{1} \]

\[ \text{moveToORPOut\_def} \]
\[ \vdash \text{moveToORPOut MOVE\_TO\_ORP (exec (SLc pltForm)) = PLTForm} \wedge \\
(\text{moveToORPOut MOVE\_TO\_ORP (exec (SLc incomplete)) = MoveToORP} \wedge \\
(\text{moveToORPOut PLT\_FORM (exec (SLc pltMove)) = PLTMove} \wedge \\
(\text{moveToORPOut PLT\_FORM (exec (SLc incomplete)) = PLTForm} \wedge \\
(\text{moveToORPOut PLT\_MOVE (exec (SLc pltSecureHalt)) = PLTSecureHalt} \wedge \\
(\text{moveToORPOut PLT\_MOVE (exec (SLc incomplete)) = PLTMove} \wedge \\
(\text{moveToORPOut PLT\_SECURE\_HALT (exec (SLc complete)) = } \\
\text{Complete} \wedge \\
(\text{moveToORPOut PLT\_SECURE\_HALT (exec (SLc incomplete)) = } \\
\text{PLTSecureHalt} \wedge \\
(\text{moveToORPOut s (trap (SLc cmd)) = unAuthorized} \wedge \\
(\text{moveToORPOut s (discard (SLc cmd)) = unAuthenticated} \wedge \\
\text{moveToORPOut\_ind} \]
\[ \vdash \forall P. \\
P \text{ MOVE\_TO\_ORP (exec (SLc pltForm)) } \wedge \\
P \text{ MOVE\_TO\_ORP (exec (SLc incomplete)) } \wedge \\
P \text{ PLT\_FORM (exec (SLc pltMove)) } \wedge \\
P \text{ PLT\_FORM (exec (SLc incomplete)) } \wedge \\
P \text{ PLT\_MOVE (exec (SLc pltSecureHalt)) } \wedge \\
P \text{ PLT\_MOVE (exec (SLc incomplete)) } \wedge \\
P \text{ PLT\_SECURE\_HALT (exec (SLc complete)) } \wedge \\
P \text{ PLT\_SECURE\_HALT (exec (SLc incomplete)) } \wedge \\
(\forall s \text{ cmd. } P \text{ s (trap (SLc cmd))} ) \wedge \\
(\forall s \text{ cmd. } P \text{ s (discard (SLc cmd))} ) \wedge \\
(\forall s \text{ v_{8}. } P \text{ s (discard (ESCc v_{8}))} ) \wedge \\
(\forall s \text{ v_{9}. } P \text{ s (trap (ESCc v_{9}))} ) \wedge \\
(\forall v_{12}. P \text{ MOVE\_TO\_ORP (exec (ESCc v_{12}))} \wedge \\
P \text{ MOVE\_TO\_ORP (exec (SLc pltMove)) } \wedge \\
P \text{ MOVE\_TO\_ORP (exec (SLc pltSecureHalt)) } \wedge \\
P \text{ MOVE\_TO\_ORP (exec (SLc complete)) } \wedge \\
(\forall v_{15}. P \text{ PLT\_FORM (exec (ESCc v_{15}))} ) \wedge \\
P \text{ PLT\_FORM (exec (SLc pltForm)) } \wedge \\
P \text{ PLT\_FORM (exec (SLc pltSecureHalt)) } \wedge \\
P \text{ PLT\_FORM (exec (SLc complete)) } \wedge \\
(\forall v_{18}. P \text{ PLT\_MOVE (exec (ESCc v_{18}))} ) \wedge \\
P \text{ PLT\_MOVE (exec (SLc pltForm)) } \wedge \\
\]
MOVETOORPTYPE THEORY

\[ P \text{ PLT\_MOVE (exec (SLc \text{pltMove}))} \land \\
P \text{ PLT\_MOVE (exec (SLc \text{complete}))} \land \\
(\forall v_{21}. P \text{ PLT\_SECURE\_HALT (exec (ESCc v_{21})))} \land \\
P \text{ PLT\_SECURE\_HALT (exec (SLc \text{pltForm}))} \land \\
P \text{ PLT\_SECURE\_HALT (exec (SLc \text{pltMove}))} \land \\
P \text{ PLT\_SECURE\_HALT (exec (SLc \text{pltSecureHalt}))} \land \\
(\forall v_{23}. P \text{ COMPLETE (exec v_{23}))} \Rightarrow \\
\forall v v_{1}. P \text{ v v}_{1} \]

\[\text{PlatoonLeader\_exec\_slCommand\_justified\_thm}\]
\[ \vdash \forall NS \ Out \ M \ Oi \ Os. \\
\text{TR} (M, Oi, Os) \ (exec (SLc sl\text{Command})) \\
\text{CFG authTestMoveToORP ssmMoveToORPStateInterp} \\
\text{(secContextMoveToORP sl\text{Command})} \\
\text{(Name PlatoonLeader says prop (SOME (SLc sl\text{Command}))::} \\
\text{ins) s outs)} \\
\text{CFG authTestMoveToORP ssmMoveToORPStateInterp} \\
\text{(secContextMoveToORP sl\text{Command}) ins} \\
\text{(NS s (exec (SLc sl\text{Command})))} \\
\text{(Out s (exec (SLc sl\text{Command})):outs))} \iff \\
\text{authTestMoveToORP} \\
\text{(Name PlatoonLeader says prop (SOME (SLc sl\text{Command})))} \land \\
\text{CFG\text{Interpret} (M, Oi, Os)} \\
\text{CFG authTestMoveToORP ssmMoveToORPStateInterp} \\
\text{(secContextMoveToORP sl\text{Command})} \\
\text{(Name PlatoonLeader says prop (SOME (SLc sl\text{Command}))::} \\
\text{ins) s outs) \land} \\
\text{(M, Oi, Os) sat prop (SOME (SLc sl\text{Command}))} \]

\[\text{PlatoonLeader\_slCommand\_lemma}\]
\[ \vdash \text{CFG\text{Interpret} (M, Oi, Os)} \\
\text{CFG authTestMoveToORP ssmMoveToORPStateInterp} \\
\text{(secContextMoveToORP sl\text{Command})} \\
\text{(Name PlatoonLeader says prop (SOME (SLc sl\text{Command}))::} \\
\text{ins) s outs) \Rightarrow} \\
\text{(M, Oi, Os) sat prop (SOME (SLc sl\text{Command}))} \]

14 MoveToORPType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

14.1 Datatypes

\( sl\text{Command} = \text{plt\text{Form}} \mid \text{plt\text{Move}} \mid \text{plt\text{Secure\_Halt}} \mid \text{complete} \mid \text{incomplete} \)
14.2 Theorems

\[ \text{slOutput} = \text{MoveToORP} \mid \text{PLTForm} \mid \text{PLTMove} \mid \text{PLTSecureHalt} \mid \text{Complete} \mid \text{unAuthorized} \mid \text{unAuthenticated} \]

\[ \text{slState} = \text{MOVE_TO_ORP} \mid \text{PLT_FORM} \mid \text{PLT_MOVE} \mid \text{PLT_SECURE_HALT} \mid \text{COMPLETE} \]

\[ \text{stateRole} = \text{PlatoonLeader} \]

15 ssmMoveToPB Theory

\textbf{Built:} 10 June 2018

\textbf{Parent Theories:} MoveToPBType, ssm11, OMNIType

15.1 Definitions

\[ \forall \text{cmd}. \]

\[ \text{secContextMoveToPB cmd} = \]

\[ \text{[Name PlatoonLeader controls prop (SOME (SLc cmd))]} \]
15.2 Theorems

[authTestMoveToPB_cmd_reject_lemma]
\[ \forall \text{cmd}. \quad \neg \text{authTestMoveToPB} (\text{prop} \ (\text{SOME cmd})) \]

[authTestMoveToPB_def]
\[ \vdash \forall \text{state}. \quad \text{ssmMoveToPBStateInterp} \text{ state} = \text{TT} \]

(authTestMoveToPB (Name PlatoonLeader says prop cmd) \iff T) \land
(authTestMoveToPB TT \iff F) \land
(authTestMoveToPB prop \iff F) \land
(authTestMoveToPB (v2 andf v3) \iff F) \land
(authTestMoveToPB (v4 eqf v9) \iff F) \land
(authTestMoveToPB (v6 eqf v7) \iff F) \land
(authTestMoveToPB (v8 eqf v9) \iff F) \land
(authTestMoveToPB (v10 says TT) \iff F) \land
(authTestMoveToPB (v10 says FF) \iff F) \land
(authTestMoveToPB (v133 meet v134 says prop v66) \iff F) \land
(authTestMoveToPB (v135 quoting v136 says prop v66) \iff F) \land
(authTestMoveToPB (v10 says notf v67) \iff F) \land
(authTestMoveToPB (v10 says (v68 andf v69)) \iff F) \land
(authTestMoveToPB (v10 says (v70 orf v71)) \iff F) \land
(authTestMoveToPB (v10 says (v72 impf v73)) \iff F) \land
(authTestMoveToPB (v10 says (v74 eqf v75)) \iff F) \land
(authTestMoveToPB (v10 says v76 says v77) \iff F) \land
(authTestMoveToPB (v10 says v78 speaks_for v79) \iff F) \land
(authTestMoveToPB (v10 says v80 controls v81) \iff F) \land
(authTestMoveToPB (v10 says reps v82 v83 v84) \iff F) \land
(authTestMoveToPB (v10 says v85 domi v86) \iff F) \land
(authTestMoveToPB (v10 says v87 eqi v88) \iff F) \land
(authTestMoveToPB (v10 says v89 doms v86) \iff F) \land
(authTestMoveToPB (v10 says v91 eqs v92) \iff F) \land
(authTestMoveToPB (v10 says v93 eqn v94) \iff F) \land
(authTestMoveToPB (v10 says v95 lte v96) \iff F) \land
(authTestMoveToPB (v10 says v97 lt v98) \iff F) \land
(authTestMoveToPB (v12 speaks_for v13) \iff F) \land
(authTestMoveToPB (v14 controls v15) \iff F) \land
(authTestMoveToPB (reps v16 v17 v18) \iff F) \land
(authTestMoveToPB (v19 domi v20) \iff F) \land
(authTestMoveToPB (v21 eqi v22) \iff F) \land
(authTestMoveToPB (v23 doms v24) \iff F) \land
(authTestMoveToPB (v25 eqs v26) \iff F) \land
(authTestMoveToPB (v27 eqn v28) \iff F) \land
(authTestMoveToPB (v29 lte v30) \iff F) \land
(authTestMoveToPB (v31 lt v32) \iff F) \land
[authTestMoveToPB_ind]

\[ \forall P. \\
(\forall cmd. P (\text{Name PlatoonLeader says prop cmd})) \land P \ TT \land \\
P \ FF \land (\forall v. P (\text{prop v})) \land (\forall v_1. P (\text{notf v}_1)) \land \\
(\forall v_2. v_3. P (v_2 \ andf v_3)) \land (\forall v_4. v_5. P (v_4 \ orf v_5)) \land \\
(\forall v_6. v_7. P (v_6 \ \text{impf v}_7)) \land (\forall v_8. v_9. P (v_8 \ \text{eqf v}_9)) \land \\
(\forall v_{10}. P (v_{10} \ \text{says TT})) \land (\forall v_{10}. P (v_{10} \ \text{says FF})) \land \\
(\forall v_{133}. v_{134} v_{96}. P (v_{133} \ \text{meet v}_{134} \ \text{says prop v}_{96})) \land \\
(\forall v_{135} v_{136} v_{96}. P (v_{135} \ \text{quoting v}_{136} \ \text{says prop v}_{96})) \land \\
(\forall v_{10} v_{97}. P (v_{10} \ \text{says notf v}_{97})) \land \\
(\forall v_{10} v_{98} v_{99}. P (v_{10} \ \text{says (v}_{98} \ andf v}_{99})) \land \\
(\forall v_{10} v_{90} v_{71}. P (v_{10} \ \text{says (v}_{90} \ \text{orf v}_{71})) \land \\
(\forall v_{10} v_{72} v_{73}. P (v_{10} \ \text{says (v}_{72} \ \text{impf v}_{73})) \land \\
(\forall v_{10} v_{74} v_{75}. P (v_{10} \ \text{says (v}_{74} \ \text{eqf v}_{75})) \land \\
(\forall v_{10} v_{76} v_{77}. P (v_{10} \ \text{says v}_{76} \ \text{says v}_{77}) \land \\
(\forall v_{10} v_{78} v_{79}. P (v_{10} \ \text{says v}_{78} \ \text{speaks_for v}_{79})) \land \\
(\forall v_{10} v_{80} v_{81}. P (v_{10} \ \text{says v}_{80} \ \text{controls v}_{81})) \land \\
(\forall v_{10} v_{82} v_{83} v_{84}. P (v_{10} \ \text{says reps v}_{82} v_{83} v_{84})) \land \\
(\forall v_{10} v_{85} v_{86}. P (v_{10} \ \text{says v}_{85} \ \text{domi v}_{86})) \land \\
(\forall v_{10} v_{87} v_{88}. P (v_{10} \ \text{says v}_{87} \ \text{eqi v}_{88})) \land \\
(\forall v_{10} v_{89} v_{90}. P (v_{10} \ \text{says v}_{89} \ \text{doms v}_{90})) \land \\
(\forall v_{10} v_{91} v_{92}. P (v_{10} \ \text{says v}_{91} \ \text{eqs v}_{92})) \land \\
(\forall v_{10} v_{93} v_{94}. P (v_{10} \ \text{says v}_{93} \ \text{eqn v}_{94})) \land \\
(\forall v_{10} v_{95} v_{96}. P (v_{10} \ \text{says v}_{95} \ \text{lte v}_{96})) \land \\
(\forall v_{10} v_{97} v_{98}. P (v_{10} \ \text{says v}_{97} \ \text{lt v}_{98})) \land \\
(\forall v_{12} v_{13}. P (v_{12} \ \text{speaks_for v}_{13})) \land \\
(\forall v_{14} v_{15}. P (v_{14} \ \text{controls v}_{15})) \land \\
(\forall v_{16} v_{17} v_{18}. P (\text{reps v}_{16} v_{17} v_{18})) \land \\
(\forall v_{19} v_{20}. P (v_{19} \ \text{domi v}_{20})) \land \\
(\forall v_{21} v_{22}. P (v_{21} \ \text{eqi v}_{22})) \land \\
(\forall v_{23} v_{24}. P (v_{23} \ \text{doms v}_{24})) \land \\
(\forall v_{25} v_{26}. P (v_{25} \ \text{eqs v}_{26})) \land (\forall v_{27} v_{28}. P (v_{27} \ \text{eqn v}_{28})) \land \\
(\forall v_{29} v_{30}. P (v_{29} \ \text{lte v}_{30}) \land (\forall v_{31} v_{32}. (v_{31} \ \text{lt v}_{32}) \Rightarrow \\
\forall v. P \ v)

\]

[moveToPBNS_def]

\[ \text{(moveToPBNS \ MOVE\_TO\_PB (exec (SLc \ pltForm))) = PLT\_FORM} \land \\
(\text{(moveToPBNS \ MOVE\_TO\_PB (exec (SLc \ incomplete))) = \\
MOVE\_TO\_PB}) \land \\
(\text{(moveToPBNS \ PLT\_FORM (exec (SLc \ pltMove))) = PLT\_MOVE}) \land \\
(\text{(moveToPBNS \ PLT\_FORM (exec (SLc \ incomplete))) = PLT\_FORM}) \land \\
(\text{(moveToPBNS \ PLT\_MOVE (exec (SLc \ pltHalt))) = PLT\_HALT}) \land \\
(\text{(moveToPBNS \ PLT\_MOVE (exec (SLc \ incomplete))) = PLT\_MOVE}) \land \\
(\text{(moveToPBNS \ PLT\_HALT (exec (SLc \ complete))) = COMPLETE}) \land \\
(\text{(moveToPBNS \ PLT\_HALT (exec (SLc \ incomplete))) = PLT\_HALT}) \land \\
(\text{(moveToPBNS \ s (trap (SLc \ cmd))) = s}) \land \\
(\text{(moveToPBNS \ s (discard (SLc \ cmd))) = s})

65
SSMMOVE TOPB THEORY

Theorems

[moveToPBNs_ind]
\[
\forall P.
  \quad P \text{ MOVE_TO_PB (exec (SLc pltForm))} \land \\
  \quad P \text{ MOVE_TO_PB (exec (SLc incomplete))} \land \\
  \quad P \text{ PLT_FORM (exec (SLc pltMove))} \land \\
  \quad P \text{ PLT_FORM (exec (SLc incomplete))} \land \\
  \quad P \text{ PLT_MOVE (exec (SLc pltHalt))} \land \\
  \quad P \text{ PLT_MOVE (exec (SLc incomplete))} \land \\
  \quad P \text{ PLT_HALT (exec (SLc complete))} \land \\
  \quad P \text{ PLT_HALT (exec (SLc incomplete))} \land \\
  \quad (\forall s \text{ cmd. } P \text{ s (trap (SLc cmd)))} \land \\
  \quad (\forall s \text{ cmd. } P \text{ s (discard (SLc cmd)))} \land \\
  \quad (\forall s v_1, P \text{ s (discare (ESCc v_1)))} \land \\
  \quad (\forall s v_1, P \text{ s (trap (ESCc v_1)))} \land \\
  \quad (\forall s v_2, P \text{ MOVE_TO_PB (exec (ESCc v_2)))} \land \\
  \quad P \text{ MOVE_TO_PB (exec (SLc pltMove))} \land \\
  \quad P \text{ MOVE_TO_PB (exec (SLc complete))} \land \\
  \quad (\forall s v_3, P \text{ PLT_FORM (exec (ESCc v_3)))} \land \\
  \quad P \text{ PLT_FORM (exec (SLc pltForm))} \land \\
  \quad P \text{ PLT_FORM (exec (SLc incomplete))} \land \\
  \quad (\forall s v_8, P \text{ PLT_MOVE (exec (ESCc v_8)))} \land \\
  \quad P \text{ PLT_MOVE (exec (SLc pltMove))} \land \\
  \quad P \text{ PLT_MOVE (exec (SLc complete))} \land \\
  \quad (\forall v_1, P \text{ PLT_HALT (exec (ESCc v_1)))} \land \\
  \quad P \text{ PLT_HALT (exec (SLc pltForm))} \land \\
  \quad P \text{ PLT_HALT (exec (SLc pltMove))} \land \\
  \quad P \text{ PLT_HALT (exec (SLc complete))} \land \\
  \quad (\forall v_2, P \text{ COMPLETE (exec (ESCc v_2))) \Rightarrow} \\
  \quad \forall v v_1, P \text{ v v_1}
\]

[moveToPBOut_def]
\[
\vdash \text{(moveToPBOut MOVE_TO_PB (exec (SLc pltForm)) = PLTForm)} \land \\
\quad \text{(moveToPBOut MOVE_TO_PB (exec (SLc incomplete)) = MoveToPB)} \land \\
\quad \text{(moveToPBOut PLT_FORM (exec (SLc pltMove)) = PLTMove)} \land \\
\quad \text{(moveToPBOut PLT_FORM (exec (SLc incomplete)) = PLTForm)} \land \\
\quad \text{(moveToPBOut PLT_MOVE (exec (SLc pltHalt)) = PLTHalt)} \land \\
\quad \text{(moveToPBOut PLT_MOVE (exec (SLc incomplete)) = PLTMove)} \land \\
\quad \text{(moveToPBOut PLT_HALT (exec (SLc complete)) = Complete)} \land \\
\quad \text{(moveToPBOut PLT_HALT (exec (SLc incomplete)) = PLTHalt)} \land \\
\quad \text{(moveToPBOut s (trap (SLc cmd)) = unauthorized)} \land \\
\quad \text{(moveToPBOut s (discard (SLc cmd)) = unAuthenticated)}
\]

[moveToPBOut_ind]
\[
\vdash \forall P.
  \quad P \text{ MOVE_TO_PB (exec (SLc pltForm))} \land \\
\]

66
\[
P \text{MOVE\_TO\_PB} (\text{exec (SLc incomplete)}) \land \\
P \text{PLT\_FORM} (\text{exec (SLc pltMove)}) \land \\
P \text{PLT\_FORM} (\text{exec (SLc incomplete)}) \land \\
P \text{PLT\_MOVE} (\text{exec (SLc pltHalt)}) \land \\
P \text{PLT\_MOVE} (\text{exec (SLc incomplete)}) \land \\
P \text{PLT\_HALT} (\text{exec (SLc complete)}) \land \\
P \text{PLT\_HALT} (\text{exec (SLc incomplete)}) \land \\
(\forall \ s \ \text{cmd} \cdot P \ s \ (\text{trap (SLc cmd)})) \land \\
(\forall \ s \ \text{cmd} \cdot P \ s \ (\text{discard (SLc cmd)})) \land \\
(\forall \ s \ v_6 \cdot P \ s \ (\text{discard (ESCc } v_6))) \land \\
(\forall \ s \ v_9 \cdot P \ s \ (\text{trap (ESCc } v_9))) \land \\
(\forall \ v_{12} \cdot P \ \text{MOVE\_TO\_PB} (\text{exec (ESCc } v_{12}))) \land \\
P \text{move\_TO\_PB} (\text{exec (SLc pltMove)}) \land \\
P \text{MOVE\_TO\_PB} (\text{exec (SLc pltHalt)}) \land \\
P \text{MOVE\_TO\_PB} (\text{exec (SLc complete)}) \land \\
(\forall \ v_{15} \cdot P \ \text{PLT\_FORM} (\text{exec (ESCc } v_{15}))) \land \\
P \text{PLT\_FORM} (\text{exec (SLc pltForm)}) \land \\
P \text{PLT\_FORM} (\text{exec (SLc pltHalt)}) \land \\
P \text{PLT\_FORM} (\text{exec (SLc complete)}) \land \\
(\forall \ v_{18} \cdot P \ \text{PLT\_MOVE} (\text{exec (ESCc } v_{18}))) \land \\
P \text{PLT\_MOVE} (\text{exec (SLc pltForm)}) \land \\
P \text{PLT\_MOVE} (\text{exec (SLc pltHalt)}) \land \\
P \text{PLT\_MOVE} (\text{exec (SLc complete)}) \land \\
(\forall \ v_{21} \cdot P \ \text{PLT\_HALT} (\text{exec (ESCc } v_{21}))) \land \\
P \text{PLT\_HALT} (\text{exec (SLc pltForm)}) \land \\
P \text{PLT\_HALT} (\text{exec (SLc pltHalt)}) \land \\
P \text{PLT\_HALT} (\text{exec (SLc complete)}) \land \\
(\forall \ v_{23} \cdot P \ \text{COMPLETE (exec } v_{23})) \Rightarrow \\
\forall \ v_{1} \cdot P \ v \ v_{1}
\]

\[\text{PlatoonLeader}_\text{execSLCommand}_\text{justified}\_thm\]
\[
\vdash \forall \ NS \ Out \ M \ Oi \ Os.
\quad \text{TR (} M, Oi, Os \text{) (exec (SLc } sl\text{Command}))}
\quad \text{CFG authTestMoveToPB ssmMoveToPBStateInterp}
\quad \text{(secContextMoveToPB } sl\text{Command})}
\quad \text{(Name PlatoonLeader says prop (SOME (SLc } sl\text{Command})))::}
\quad \text{ins} \ s \ outs)
\quad \text{CFG authTestMoveToPB ssmMoveToPBStateInterp}
\quad \text{(secContextMoveToPB } sl\text{Command}) \text{ ins}
\quad \text{(NS s (exec (SLc } sl\text{Command}))}
\quad \text{(Out s (exec (SLc } sl\text{Command})))::outs)) \iff authTestMoveToPB}
\quad \text{(Name PlatoonLeader says prop (SOME (SLc } sl\text{Command}))) \land}
\quad \text{CFGInterpret (} M, Oi, Os \text{)}
\quad \text{CFG authTestMoveToPB ssmMoveToPBStateInterp}
\quad \text{(secContextMoveToPB } sl\text{Command})}
\quad \text{(Name PlatoonLeader says prop (SOME (SLc } sl\text{Command})))::}
\quad \text{ins} \ s \ outs) \land \\
\quad (M, Oi, Os) \text{ sat prop (SOME (SLc } sl\text{Command}))}
[PlatoonLeader_slCommand_lemma]

\[\text{CFGInterpret} \ (M, O_i, O_s)\]

\[\text{(CFG authTestMoveToPB ssmMoveToPBStateInterp} \]

\[\text{(secContextMoveToPB slCommand)}\]

\[\text{(Name PlatoonLeader says prop (SOME (SLc slCommand)))::} \]

\[\text{ins} \ s \ \text{outs) } \Rightarrow \]

\[\text{(M, O_i, O_s) sat prop (SOME (SLc slCommand))} \]

16 MoveToPBType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

16.1 Datatypes

\[\text{slCommand} = \text{pltForm} | \text{pltMove} | \text{pltHalt} | \text{complete} | \text{incomplete} \]

\[\text{slOutput} = \text{MoveToPB} | \text{PLTForm} | \text{PLTMove} | \text{PLTHalt} | \text{Complete} \]

\[\text{| unAuthorized} | \text{unAuthenticated} \]

\[\text{slState} = \text{MOVE_TO_PB} | \text{PLT_FORM} | \text{PLT_MOVE} | \text{PLT_HALT} | \text{COMPLETE} \]

\[\text{stateRole} = \text{PlatoonLeader} \]

16.2 Theorems

[slCommand_distinct_clauses]

\[\vdash \text{pltForm} \neq \text{pltMove} \land \text{pltForm} \neq \text{pltHalt} \land \text{pltForm} \neq \text{complete} \land \]

\[\text{pltForm} \neq \text{incomplete} \land \text{pltMove} \neq \text{pltHalt} \land \]

\[\text{pltMove} \neq \text{complete} \land \text{pltMove} \neq \text{incomplete} \land \]

\[\text{pltHalt} \neq \text{complete} \land \text{pltHalt} \neq \text{incomplete} \land \]

\[\text{complete} \neq \text{incomplete} \]

[slOutput_distinct_clauses]

\[\vdash \text{MoveToPB} \neq \text{PLTForm} \land \text{MoveToPB} \neq \text{PLTMove} \land \]

\[\text{MoveToPB} \neq \text{PLTHalt} \land \text{MoveToPB} \neq \text{Complete} \land \]

\[\text{MoveToPB} \neq \text{unAuthorized} \land \text{MoveToPB} \neq \text{unAuthenticated} \land \]

\[\text{PLTForm} \neq \text{PLTMove} \land \text{PLTForm} \neq \text{PLTHalt} \land \text{PLTForm} \neq \text{Complete} \land \]

\[\text{PLTForm} \neq \text{unAuthorized} \land \text{PLTForm} \neq \text{unAuthenticated} \land \]

\[\text{PLTMove} \neq \text{PLTHalt} \land \text{PLTMove} \neq \text{Complete} \land \]

\[\text{PLTMove} \neq \text{unAuthorized} \land \text{PLTMove} \neq \text{unAuthenticated} \land \]

\[\text{PLTHalt} \neq \text{Complete} \land \text{PLTHalt} \neq \text{unAuthorized} \land \]

\[\text{PLTHalt} \neq \text{unAuthenticated} \land \text{Complete} \neq \text{unAuthorized} \land \]

\[\text{Complete} \neq \text{unAuthenticated} \land \text{unAuthorized} \neq \text{unAuthenticated} \]

68
\[\text{slState\_distinct\_clauses}\]

\[\text{MOVE\_TO\_PB \neq PLT\_FORM \land MOVE\_TO\_PB \neq PLT\_MOVE \land MOVE\_TO\_PB \neq PLT\_HALT \land MOVE\_TO\_PB \neq COMPLETE} \land
\text{PLT\_FORM \neq PLT\_MOVE \land PLT\_FORM \neq PLT\_HALT}\land
\text{PLT\_FORM \neq COMPLETE \land PLT\_MOVE \neq PLT\_HALT}\land
\text{PLT\_MOVE \neq COMPLETE \land PLT\_HALT \neq COMPLETE}\]

17 ssPlanPB Theory

Built: 10 June 2018

Parent Theories: PlanPBDef, ssm

17.1 Theorems

[inputOK\_def]

\[\text{(inputOK (Name PlatoonLeader says prop cmd) } \iff \text{T)} \land
\text{(inputOK (Name PlatoonSergeant says prop cmd) } \iff \text{T)} \land
\text{(inputOK TT } \iff \text{F) } \land
\text{(inputOK (prop } v) \iff \text{F) } \land
\text{(inputOK (notf } v) \iff \text{F) } \land
\text{(inputOK (v \ andf v) } \iff \text{F) } \land
\text{(inputOK (v \ impf v) } \iff \text{F) } \land
\text{(inputOK (v \ eqf v) } \iff \text{F) } \land
\text{(inputOK (v \ domi v) } \iff \text{F) } \land
\text{(inputOK (v \ eqi v) } \iff \text{F) } \land
\text{(inputOK (v \ doms v) } \iff \text{F) } \land
\text{(inputOK (v \ eqn v) } \iff \text{F) } \land
\text{(inputOK (v \ lte v) } \iff \text{F) } \land
\text{(inputOK (v \ lt v) } \iff \text{F) } \land
\text{(inputOK (v10 says v13 says prop v66) } \iff \text{F) } \land
\text{(inputOK (v135 quoting v136 says prop v66) } \iff \text{F) } \land
\text{(inputOK (v10 says notf v97) } \iff \text{F) } \land
\text{(inputOK (v10 says (v68 andf v69)) } \iff \text{F) } \land
\text{(inputOK (v10 says (v70 orf v71)) } \iff \text{F) } \land
\text{(inputOK (v10 says (v72 impf v73)) } \iff \text{F) } \land
\text{(inputOK (v10 says (v74 eqf v75)) } \iff \text{F) } \land
\text{(inputOK (v10 says v76 says v77) } \iff \text{F) } \land
\text{(inputOK (v10 says v78 speaks_for v79) } \iff \text{F) } \land
\text{(inputOK (v10 says v80 controls v81) } \iff \text{F) } \land
\text{(inputOK (v10 says reps v82 v83 v84) } \iff \text{F) } \land
\text{(inputOK (v10 says v95 domi v96) } \iff \text{F) } \land
\text{(inputOK (v10 says v97 eqi v98) } \iff \text{F) } \land
\text{(inputOK (v10 says v99eqs v98) } \iff \text{F) } \land
\text{(inputOK (v10 says v99 eqn v95) } \iff \text{F) } \land
\text{(inputOK (v10 says v99 lte v96) } \iff \text{F) } \land
\text{(inputOK (v10 says v97 lt v98) } \iff \text{F) } \land
\text{(inputOK (v12 speaks_for v13) } \iff \text{F) } \land
\text{(inputOK (v14 controls v15) } \iff \text{F) } \land
\text{(inputOK (reps v16 v17 v18) } \iff \text{F) } \land
\text{(inputOK (v19 domi v20) } \iff \text{F) } \land
\text{(inputOK (v21 eqi v22) } \iff \text{F) } \land
\text{(inputOK (v23 doms v24) } \iff \text{F) } \land
\( \text{inputOK} (v_{25} \text{ eqs } v_{26}) \iff F \land (\text{inputOK} (v_{27} \text{ eqn } v_{28}) \iff F) \land \\
(\text{inputOK} (v_{29} \text{ lte } v_{30}) \iff F) \land (\text{inputOK} (v_{31} \text{ lt } v_{32}) \iff F) \)

**Theorems**

\[ \vdash \forall P. \]
\[ (\forall \text{cmd}. \ P (\text{Name PlatoonLeader says prop cmd})) \land \\
(\forall \text{cmd}. \ P (\text{Name PlatoonSergeant says prop cmd})) \land P \text{ TT} \land \\
P \text{ FF} \land (\forall v. \ P (\text{prop } v)) \land (\forall v_1. \ P (\text{notf } v_1)) \land \\
(\forall v_2 v_3. \ P (v_2 \text{ andf } v_3)) \land (\forall v_4 v_5. \ P (v_4 \text{ orf } v_5)) \land \\
(\forall v_6 v_7. \ P (v_6 \text{ impf } v_7)) \land (\forall v_8 v_9. \ P (v_8 \text{ eqf } v_9)) \land \\
(\forall v_{10}. \ P (v_{10} \text{ says TT})) \land (\forall v_{10}. \ P (v_{10} \text{ says FF})) \land \\
(\forall v_{133} v_{134} v_{96}. \ P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{96})) \land \\
(\forall v_{135} v_{136} v_{96}. \ P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{96})) \land \\
(\forall v_{10} v_{97}. \ P (v_{10} \text{ says notf } v_{97})) \land \\
(\forall v_{10} v_{98} v_{99}. \ P (v_{10} \text{ says } (v_{98} \text{ andf } v_{99}))) \land \\
(\forall v_{10} v_{70} v_{71}. \ P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \land \\
(\forall v_{10} v_{72} v_{73}. \ P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \land \\
(\forall v_{10} v_{74} v_{75}. \ P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \land \\
(\forall v_{10} v_{76} v_{77}. \ P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \land \\
(\forall v_{10} v_{78} v_{79}. \ P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \land \\
(\forall v_{10} v_{80} v_{81}. \ P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \land \\
(\forall v_{10} v_{82} v_{83} v_{84}. \ P (v_{10} \text{ says } v_{82} \text{ eqs } v_{83} \text{ eqs } v_{84})) \land \\
(\forall v_{10} v_{85} v_{86}. \ P (v_{10} \text{ says } v_{85} \text{ dom } v_{86})) \land \\
(\forall v_{10} v_{87} v_{88}. \ P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \land \\
(\forall v_{10} v_{89} v_{90}. \ P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \land \\
(\forall v_{10} v_{91} v_{92}. \ P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \land \\
(\forall v_{10} v_{93} v_{94}. \ P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \land \\
(\forall v_{10} v_{95} v_{96}. \ P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \land \\
(\forall v_{10} v_{97} v_{98}. \ P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \land \\
(\forall v_{12} v_{13}. \ P (v_{12} \text{ speaks_for } v_{13})) \land \\
(\forall v_{14} v_{15}. \ P (v_{14} \text{ controls } v_{15})) \land \\
(\forall v_{16} v_{17} v_{18}. \ P (\text{reps } v_{16} \text{ repl } v_{17} \text{ repl } v_{18})) \land \\
(\forall v_{19} v_{20}. \ P (v_{19} \text{ dom } v_{20})) \land \\
(\forall v_{21} v_{22}. \ P (v_{21} \text{ eqi } v_{22})) \land \\
(\forall v_{23} v_{24}. \ P (v_{23} \text{ doms } v_{24})) \land \\
(\forall v_{25} v_{26}. \ P (v_{25} \text{ eqs } v_{26})) \land (\forall v_{27} v_{28}. \ P (v_{27} \text{ eqn } v_{28})) \land \\
(\forall v_{29} v_{30}. \ P (v_{29} \text{ lte } v_{30})) \land (\forall v_{31} v_{32}. \ P (v_{31} \text{ lt } v_{32}) \Rightarrow \\
\forall v. \ P v \\
\]

**planPBNS_def**

\[ \vdash (\text{planPBNS} \text{ WARNO} (\text{exec } x) = \\
\text{if} \ \\
(\text{getRecon } x = \text{[SOME (SCLc (PL recon))]})) \land \\
(\text{getTentativePlan } x = \text{[SOME (SCLc (PL tentativePlan))]})) \land \\
(\text{getReport } x = \text{[SOME (SCLc (PL reporti))]})) \land \\
(\text{getInitMove } x = \text{[SOME (SCLc (PSG initiateMovement))]})) \land \\
\text{then} \\
\text{REPORT1} \\
\text{else WARNO} \land \]

70
(planPBNS PLAN_PB (exec x) =
  if getPlCom x = receiveMission then RECEIVE_MISSION
  else PLAN_PB) ∧
(planPBNS RECEIVE_MISSION (exec x) =
  if getPlCom x = warno then WARNO else RECEIVE_MISSION) ∧
(planPBNS REPORT1 (exec x) =
  if getPlCom x = completePlan then COMPLETE_PLAN
  else REPORT1) ∧
(planPBNS COMPLETE_PLAN (exec x) =
  if getPlCom x = opoid then OPOID else COMPLETE_PLAN) ∧
(planPBNS OPOID (exec x) =
  if getPlCom x = supervise then SUPERVISE else OPOID) ∧
(planPBNS REPORT2 (exec x) =
  if getPlCom x = complete then COMPLETE else REPORT2) ∧
(planPBNS s (trap v0) = s) ∧ (planPBNS s (discard v1) = s)

[planPBNS_ind]
∀ x. P 
  (∀ x. P WARNO (exec x)) ∧ (∀ x. P PLAN_PB (exec x)) ∧
  (∀ x. P RECEIVE_MISSION (exec x)) ∧
  (∀ x. P REPORT1 (exec x)) ∧ (∀ x. P COMPLETE_PLAN (exec x)) ∧
  (∀ x. P OPOID (exec x)) ∧ (∀ x. P SUPERVISE (exec x)) ∧
  (∀ x. P REPORT2 (exec x)) ∧ (∀ x. P COMPLETE (exec x))

[planPBOut_def]
∀ x. P WARNO (exec x) =
  if (getRecon x = [SOME (SLc (PL recon))]) ∧
  (getTentativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
  (getReport x = [SOME (SLc (PL report1))]) ∧
  (getInitMove x = [SOME (SLc (PSG initiateMovement))])
  then Report1
  else unAuthorized) ∧
(planPBOut PLAN_PB (exec x) =
  if getPlCom x = receiveMission then ReceiveMission
  else unAuthorized) ∧
(planPBOut RECEIVE_MISSION (exec x) =
  if getPlCom x = warno then Warno else unAuthorized) ∧
(planPBOut REPORT1 (exec x) =
  if getPlCom x = completePlan then CompletePlan
  else unAuthorized) ∧
(planPBOut COMPLETE_PLAN (exec x) =
  if getPlCom x = opioid then Opid else unAuthorized) ∧
(planPBOut OPOID (exec x) =
  if getPlCom x = supervise then Supervise
  else unAuthorized) ∧
(planPBOut SUPERVISE (exec x) =
  if getPlCom x = report2 then Report2 else unAuthorized) ∧
(planPBOut REPORT2 (exec x) =
  if getPlCom x = complete then Complete else unAuthorized) ∧
(planPBOut s (trap v0) = unAuthorized) ∧
(planPBOut s (discard v1) = unAuthenticated)

[planPBOut_ind]
\[\forall P.
  (\forall x. P \text{ WARNO (exec } x) \land (\forall x. P \text{ PLAN_PB (exec } x)) \land
  (\forall x. P \text{ RECEIVE MISSION (exec } x)) \land
  (\forall x. P \text{ REPORT1 (exec } x)) \land (\forall x. P \text{ COMPLETE_PLAN (exec } x)) \land
  (\forall x. P \text{ OPOID (exec } x)) \land (\forall x. P \text{ SUPERVISE (exec } x)) \land
  (\forall x. P \text{ REPORT2 (exec } x)) \land (\forall s v0. P s (\text{trap } v0)) \land
  (\forall s v1. P s (\text{discard } v1)) \land
  (\forall v6. P \text{ TENTATIVE_PLAN (exec } v6)) \land
  (\forall v7. P \text{ INITIATE_MOVEMENT (exec } v7)) \land
  (\forall v8. P \text{ RECON (exec } v8)) \land (\forall v9. P \text{ COMPLETE (exec } v9)) \Rightarrow
  \forall v. P \text{ v6 } \text{v7 } \text{v8 } \text{v9 })

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]
\[\forall s \neq \text{WARNO} \Rightarrow
  \text{plCommand} \neq \text{invalidPlCommand} \Rightarrow
  \text{plCommand} \neq \text{report1} \Rightarrow
  \forall NS \text{ Out } M \text{ Oi Os.}
  \text{TR (M,Oi,Os)}
  (\text{exec}
    \text{(inputList}
      [\text{Name PlatoonLeader says}
        \text{prop (SOME (SLc (PL plCommand))))})
    \text{(CFG inputOK secContext secContextNull}
      ([\text{Name PlatoonLeader says}
        \text{prop (SOME (SLc (PL plCommand)))]::ins) s outs})
    \text{(CFG inputOK secContext secContextNull ins}
      (NS s)
    \text{exec}
      \text{(inputList}
        [\text{Name PlatoonLeader says}
          \text{prop (SOME (SLc (PL plCommand))))})
    \text{(Out s}
    \text{exec}
      \text{(inputList}
        [\text{Name PlatoonLeader says}
          \text{prop (SOME (SLc (PL plCommand))))})::

72
authenticationTest inputOK
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))] \land
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secContextNull
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))])::ins) s outs) \land
(M, Oi, Os) satList
propCommandList
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))]

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]
\[ s \neq \text{WARNO} \implies
\begin{align*}
\text{plCommand} & \neq \text{invalidPlCommand} \implies \\
\text{plCommand} & \neq \text{report1} \implies \\
\forall M Oi Os. \\
\text{TR} (M, Oi, Os) \ (\text{exec} \ [\text{SOME} (\text{SLc} (\text{PL plCommand})))) \\
& (\text{CFG inputOK secContext secContextNull}
[\text{Name PlatoonLeader says}
  \text{prop (SOME (SLc (PL plCommand)))}])::ins) s outs) \\
& (\text{CFG inputOK secContext secContextNull ins}
\ \text{(NS s (exec [SOME (SLc (PL plCommand))))}) \\
& (\text{Out s (exec [SOME (SLc (PL plCommand))])}::outs)) \iff
\end{align*}
\text{authenticationTest inputOK}
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))] \land
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secContextNull
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))])::ins) s outs) \land
(M, Oi, Os) satList [prop (SOME (SLc (PL plCommand)))]

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]
\[ s \neq \text{WARNO} \implies
\begin{align*}
\text{plCommand} & \neq \text{invalidPlCommand} \implies \\
\text{plCommand} & \neq \text{report1} \implies \\
\forall M Oi Os. \\
\text{CFGInterpret} (M, Oi, Os)
\ (\text{CFG inputOK secContext secContextNull}
\ [\text{Name PlatoonLeader says}
  \text{prop (SOME (SLc (PL plCommand)))}])::ins) s outs) \implies \\
(M, Oi, Os) satList \ [\text{prop (SOME (SLc (PL plCommand)))]}
propCommandList
[Name PlatoonLeader says
  prop (SOME (SLc (PL plCommand)))]
\[\text{PlatoonLeader\_psgCommand\_notDiscard\_thm}\]
\[\forall NS \ Out M \ Oi \ Os.
\neg TR (M, Oi, Os) \ (\text{discard} [\text{SOME} (\text{SLc (PSG \ psgCommand))}])
\]
\[
\begin{align*}
(CFG \ \text{inputOK} & \ \text{secContext} \ \text{secContextNull} \\
& ([\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]): ins \ s \ outs) \\
& (CFG \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \ ins \\
& (NS \ s (\text{discard} [\text{SOME} (\text{SLc (PSG \ psgCommand))}]))) \\
& (Out s (\text{discard} [\text{SOME} (\text{SLc (PSG \ psgCommand)}))]): \ outs))
\end{align*}
\]

\[\text{PlatoonLeader\_trap\_psgCommand\_justified\_lemma}\]
\[\forall NS \ Out M \ Oi \ Os.
TR (M, Oi, Os) \\
\begin{align*}
& (\text{trap} \\
& (\text{inputList} \\
& \ \ [\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]))) \\
& (CFG \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \\
& ([\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]): ins \ s \ outs) \\
& (CFG \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \ ins \\
& (NS \ s \\
& \begin{align*}
& (\text{trap} \\
& (\text{inputList} \\
& \ \ [\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]))) \\
& (Out s \\
& \begin{align*}
& (\text{trap} \\
& (\text{inputList} \\
& \ \ [\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]))) \implies \\
& \text{authenticationTest} \ \text{inputOK} \\
& \ [\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]) \land \\
& \ \text{CFGInterpret} (M, Oi, Os) \\
& (CFG \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \\
& ([\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]): ins \ s \ outs) \land \\
& (M, Oi, Os) \ sat \ prop \ \text{NONE})
\end{align*}
\end{align*}
\]

\[\text{PlatoonLeader\_trap\_psgCommand\_lemma}\]
\[\forall M \ Oi \ Os.
\ \text{CFGInterpret} (M, Oi, Os) \\
\begin{align*}
& (CFG \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \\
& ([\text{Name PlatoonLeader says} \\
& \ \ prop (\text{SOME} (\text{SLc (PSG \ psgCommand)}))]): ins \ s \ outs) \Rightarrow \\
& (M, Oi, Os) \ sat \ prop \ \text{NONE}
\end{align*}
\]

74
[PlatoonLeader_WARNO_exec_report1_justified_lemma]
\[ \forall NS Out M Oi Os. \]
\[ \text{TR} (M, Oi, Os) \]
\[ \text{(exec} \]
\[ \text{inputList} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan))));} \]
\[ \text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))))));} \]
\[ \text{(CFG inputOK secContext secContextNull} \]
\[ \text{(Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan))));} \]
\[ \text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))))));} \]
\[ \text{(NS WARNO secContext secContextNull ins} \]
\[ \text{(exec} \]
\[ \text{inputList} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan))));} \]
\[ \text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))))));} \]
\[ \text{(Out WARNO} \]
\[ \text{(exec} \]
\[ \text{inputList} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan))));} \]
\[ \text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL report1))))));} \]
\[ \text{authenticationTest inputOK} \]
\[ \text{[Name PlatoonLeader says prop (SOME (SLc (PL recon))));} \]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan))));} \]
Name PlatoonSergeant says
prop (SOME (SLc (PSG initiateMovement)));
Name PlatoonLeader says
prop (SOME (S1c (PL report1)));

CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secContextNull
(\([\text{Name PlatoonLeader says prop (SOME (S1c (PL recon))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL tentativePlan))});
\text{Name PlatoonSergeant says prop (SOME (S1c (PSG initiateMovement))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL report1))))\])::ins) WARNO outs) ∧
(M, Oi, Os) satList
propCommandList
\([\text{Name PlatoonLeader says prop (SOME (S1c (PL recon))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL tentativePlan))});
\text{Name PlatoonSergeant says prop (SOME (S1c (PSG initiateMovement))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL report1))))\])]}8

\[\text{PlatoonLeader\_WARNO\_exec\_report1\_justified\_thm}\]

\(\forall NS Out M Oi Os.\)
\(\text{TR} (M, Oi, Os)\)
\((\text{exec} [\text{SOME (S1c (PL recon))}; \text{SOME (S1c (PL tentativePlan))};\)
\text{SOME (S1c (PSG initiateMovement))};\)
\text{SOME (S1c (PL report1))])\)
\(\text{CFG inputOK secContext secContextNull}\)
\(([\text{Name PlatoonLeader says prop (SOME (S1c (PL recon))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL tentativePlan))});
\text{Name PlatoonSergeant says prop (SOME (S1c (PSG initiateMovement))});
\text{Name PlatoonLeader says prop (SOME (S1c (PL report1))))\])::ins) WARNO outs)\)
\(\text{CFG inputOK secContext secContextNull ins}\)
\(\text{NS WARNO}\)
\((\text{exec} [\text{SOME (S1c (PL recon))};\)
\text{SOME (S1c (PL tentativePlan))};\)
\text{SOME (S1c (PSG initiateMovement))};\)
\text{SOME (S1c (PL report1))])\)
\(\text{Out WARNO}\)
\((\text{exec} [\text{SOME (S1c (PL recon))});
Theorems

SSMPLANPB THEORY

\[
\begin{align*}
\text{authenticationTest inputOK} & \\
[\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan)))}; \\
\text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))}] \\
\text{CFGInterpret} (M, Oi, Os) & \\
\left(\text{CFG inputOK secContext secContextNull} \\
\quad \left(\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan)))}; \\
\text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))}]::ins\right) \text{ WARNO outs} \right) & \\
(M, Oi, Os) & \text{satList} \\
\text{propCommandList} & \\
[\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan)))}; \\
\text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))}] & \\
\text{PlatoonLeader_WARNO_exec_report1_lemma} \quad [\text{LEMMA}] & \\
\vdash \forall M \, Oi \, Os. \\
\text{CFGInterpret} (M, Oi, Os) & \\
\left(\text{CFG inputOK secContext secContextNull} \\
\quad \left(\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan)))}; \\
\text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))}]::ins\right) \text{ WARNO outs} \right) \Rightarrow \\
(M, Oi, Os) & \text{satList} \\
\text{propCommandList} & \\
[\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL tentativePlan)))}; \\
\text{Name PlatoonSergeant says prop (SOME (SLc (PSG initiateMovement)))}; \\
\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))}] & \\
\text{PlatoonSergeant_trap_plCommand_justified_lemma} \quad [\text{LEMMA}] &
\end{align*}
\]
\[\forall NS \ Out M Oi Os.
\begin{align*}
    &\text{TR} (M, Oi, Os) \\
    &\text{(trap)} \\
    &\text{inputList} \\
    &\text{[Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]}) \\
    &\text{(CFG inputOK secContext secContextNull)} \\
    &\text{([Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]::ins) s outs} \\
    &\text{(CFG inputOK secContext secContextNull ins)} \\
    &\text{(NS s)} \\
    &\text{(trap)} \\
    &\text{inputList} \\
    &\text{[Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]}) \\
    &\text{(Out s)} \\
    &\text{(trap)} \\
    &\text{inputList} \\
    &\text{[Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]})::outs) \\
\end{align*}
\]

\[\text{authenticationTest inputOK} \\
\text{[Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]} \land \\
\text{CFGInterpret (M, Oi, Os)} \\
\text{([Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]::ins) s outs} \land \\
\text{(M, Oi, Os) sat prop NONE}
\]

\[\text{PlatoonSergeant_trap_plCommand_justified_thm}\]

\[\forall NS \ Out M Oi Os.
\begin{align*}
    &\text{TR} (M, Oi, Os) (\text{trap [SOME (SLc (PL plCommand))]}) \\
    &\text{(CFG inputOK secContext secContextNull)} \\
    &\text{([Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]::ins) s outs} \\
    &\text{(CFG inputOK secContext secContextNull ins)} \\
    &\text{(NS s (trap [SOME (SLc (PL plCommand))]))} \\
    &\text{(Out s (trap [SOME (SLc (PL plCommand))]:::outs))} \\
\end{align*}
\]

\[\text{authenticationTest inputOK} \\
\text{[Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]} \land \\
\text{CFGInterpret (M, Oi, Os)} \\
\text{([Name PlatoonSergeant says prop (SOME (SLc (PL plCommand)))]::ins) s outs} \land \\
\text{(M, Oi, Os) sat prop NONE}
\]

\[\text{PlatoonSergeant_trap_plCommand_lemma}\]
\[ \forall M, O_i, O_s. \\
\text{CFGInterpret} \left( M, O_i, O_s \right) \\
\text{(CFG inputOK secContext secContextNull} \\
\text{[Name PlatoonSergeant says} \\
\text{prop (SOME (SLc (PL plCommand)))::ins) s outs) } \Rightarrow \\
\text{(M, O_i, O_s) sat prop NONE} \]

18 PlanPBType Theory

Built: 10 June 2018
Parent Theories: indexedLists, patternMatches

18.1 Datatypes

\( plCommand \) = receiveMission | warno | tentativePlan | recon \\
\quad \mid report1 | completePlan | opoid | supervise | report2 \\
\quad \mid complete | plIncomplete | invalidPlCommand

\( psgCommand \) = initiateMovement | psgIncomplete \\
\quad \mid invalidPsgCommand

\( slCommand \) = PL plCommand | PSG psgCommand

\( slOutput \) = PlanPB | ReceiveMission | Warno | TentativePlan \\
\quad \mid InitiateMovement | Recon | Report1 | CompletePlan \\
\quad \mid Opoid | Supervise | Report2 | Complete \\
\quad \mid unAuthenticated | unAuthorized

\( slState \) = PLAN_PB | RECEIVE_MISSION | WARNO | TENTATIVE_PLAN \\
\quad \mid INITIATE_MOVEMENT | RECON | REPORT1 | COMPLETE_PLAN \\
\quad \mid OPOID | SUPERVISE | REPORT2 | COMPLETE

\( stateRole \) = PlatoonLeader | PlatoonSergeant

18.2 Theorems

\([plCommand\_distinct\_clauses]\)

\[ \vdash \text{receiveMission} \neq \text{warno} \land \text{receiveMission} \neq \text{tentativePlan} \land \\
\text{receiveMission} \neq \text{recon} \land \text{receiveMission} \neq \text{report1} \land \\
\text{receiveMission} \neq \text{completePlan} \land \text{receiveMission} \neq \text{opoid} \land \\
\text{receiveMission} \neq \text{supervise} \land \text{receiveMission} \neq \text{report2} \land \\
\text{receiveMission} \neq \text{complete} \land \text{receiveMission} \neq \text{plIncomplete} \land \\
\text{receiveMission} \neq \text{invalidPlCommand} \land \text{warno} \neq \text{tentativePlan} \land \\
\text{warno} \neq \text{recon} \land \text{warno} \neq \text{report1} \land \text{warno} \neq \text{completePlan} \land \\
\text{warno} \neq \text{opoid} \land \text{warno} \neq \text{supervise} \land \text{warno} \neq \text{report2} \land \\
\text{warno} \neq \text{complete} \land \text{warno} \neq \text{plIncomplete} \land \\
\text{warno} \neq \text{invalidPlCommand} \land \text{tentativePlan} \neq \text{recon} \land \\
\text{tentativePlan} \neq \text{report1} \land \text{tentativePlan} \neq \text{completePlan} \land \\
\]
tentativePlan \neq \text{opoid} \land \text{tentativePlan} \neq \text{supervise} \land
\text{tentativePlan} \neq \text{report2} \land \text{tentativePlan} \neq \text{complete} \land
\text{tentativePlan} \neq \text{plIncomplete} \land
\text{tentativePlan} \neq \text{invalidPlCommand} \land \text{recon} \neq \text{report1} \land
\text{recon} \neq \text{completePlan} \land \text{recon} \neq \text{opoid} \land \text{recon} \neq \text{supervise} \land
\text{recon} \neq \text{report2} \land \text{recon} \neq \text{complete} \land \text{recon} \neq \text{plIncomplete} \land
\text{recon} \neq \text{invalidPlCommand} \land \text{report1} \neq \text{completePlan} \land
\text{report1} \neq \text{opoid} \land \text{report1} \neq \text{supervise} \land \text{report1} \neq \text{report2} \land
\text{report1} \neq \text{complete} \land \text{report1} \neq \text{plIncomplete} \land
\text{report1} \neq \text{invalidPlCommand} \land \text{completePlan} \neq \text{opoid} \land
\text{completePlan} \neq \text{supervise} \land \text{completePlan} \neq \text{report2} \land
\text{completePlan} \neq \text{complete} \land \text{completePlan} \neq \text{plIncomplete} \land
\text{completePlan} \neq \text{invalidPlCommand} \land \text{opoid} \neq \text{supervise} \land
\text{opoid} \neq \text{report2} \land \text{opoid} \neq \text{complete} \land \text{opoid} \neq \text{plIncomplete} \land
\text{opoid} \neq \text{invalidPlCommand} \land \text{supervise} \neq \text{report2} \land
\text{supervise} \neq \text{complete} \land \text{supervise} \neq \text{plIncomplete} \land
\text{supervise} \neq \text{invalidPlCommand} \land \text{report2} \neq \text{complete} \land
\text{report2} \neq \text{plIncomplete} \land \text{report2} \neq \text{invalidPlCommand} \land
\text{complete} \neq \text{plIncomplete} \land \text{complete} \neq \text{invalidPlCommand} \land
\text{plIncomplete} \neq \text{invalidPlCommand}

\text{[psgCommand\_distinct\_clauses]}
\vdash \text{initiateMovement} \neq \text{psgIncomplete} \land
\text{initiateMovement} \neq \text{invalidPsgCommand} \land
\text{psgIncomplete} \neq \text{invalidPsgCommand}

\text{[slCommand\_distinct\_clauses]}
\vdash \forall a'. \text{PL} a \neq \text{PSG} a'

\text{[slCommand\_one\_one]}
\vdash (\forall a'. (\text{PL} a = \text{PL} a') \iff (a = a')) \land
\forall a'. (\text{PSG} a = \text{PSG} a') \iff (a = a')

\text{[slOutput\_distinct\_clauses]}
\vdash \text{PlanPB} \neq \text{ReceiveMission} \land \text{PlanPB} \neq \text{Warno} \land
\text{PlanPB} \neq \text{TentativePlan} \land \text{PlanPB} \neq \text{InitiateMovement} \land
\text{PlanPB} \neq \text{Recon} \land \text{PlanPB} \neq \text{Report1} \land \text{PlanPB} \neq \text{CompletePlan} \land
\text{PlanPB} \neq \text{Opoid} \land \text{PlanPB} \neq \text{Supervise} \land \text{PlanPB} \neq \text{Report2} \land
\text{PlanPB} \neq \text{Complete} \land \text{PlanPB} \neq \text{unAuthenticated} \land
\text{PlanPB} \neq \text{unAuthorized} \land \text{ReceiveMission} \neq \text{Warno} \land
\text{ReceiveMission} \neq \text{TentativePlan} \land
\text{ReceiveMission} \neq \text{InitiateMovement} \land \text{ReceiveMission} \neq \text{Recon} \land
\text{ReceiveMission} \neq \text{Report1} \land \text{ReceiveMission} \neq \text{CompletePlan} \land
\text{ReceiveMission} \neq \text{Opoid} \land \text{ReceiveMission} \neq \text{Supervise} \land
\text{ReceiveMission} \neq \text{Report2} \land \text{ReceiveMission} \neq \text{Complete} \land
\text{ReceiveMission} \neq \text{unAuthenticated} \land
\text{ReceiveMission} \neq \text{unAuthorized} \land \text{Warno} \neq \text{TentativePlan} \land
\text{Warno} \neq \text{InitiateMovement} \land \text{Warno} \neq \text{Recon} \land \text{Warno} \neq \text{Report1} \land
Theorems

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]

\[ \text{planpbtype theory} \]
19 PlanPBDef Theory

Built: 10 June 2018
Parent Theories: PlanPBType, aclfoundation, OMNIType

19.1 Definitions

[PL_notWARNO_Auth_def]
\[ PL_{\text{notWARNO Auth}} \ \text{cmd} = \]
\[ \text{if cmd = report1 then prop NONE}\]
\[ \text{else}\]
\[ \text{Name PlatoonLeader says prop (SOME (SLc (PL cmd))) impf}\]
\[ \text{Name PlatoonLeader controls prop (SOME (SLc (PL cmd)))}\]

[PL_WARNO_Auth_def]
\[ PL_{\text{WARNO Auth}} = \]
\[ \text{prop (SOME (SLc (PL recon))) impf}\]
\[ \text{prop (SOME (SLc (PL tentativePlan))) impf}\]
\[ \text{prop (SOME (SLc (PSG initiateMovement))) impf}\]
\[ \text{Name PlatoonLeader controls prop (SOME (SLc (PL report1)))}\]

[secContext_def]
\[ \forall s x. \]
\[ \text{secContext s x} = \]
\[ \text{if s = WARNO then}\]
if
(getRecon x = [SOME (SLc (PL recon))]) ∧
(getTentativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
(getReport x = [SOME (SLc (PL report1))]) ∧
(getInitMove x = [SOME (SLc (PSG initiateMovement))])
then
[PL_WARNO_Auth;
    Name PlatoonLeader controls
    prop (SOME (SLc (PL recon)));
    Name PlatoonLeader controls
    prop (SOME (SLc (PL tentativePlan)));
    Name PlatoonSergeant controls
    prop (SOME (SLc (PSG initiateMovement)))]
else [prop NONE]
else if
getPlCom x = invalidPlCommand
then [prop NONE]
else [PL_notWARNO_Auth (getPlCom x)]

[secContextNull_def]
⊢ ∀x. secContextNull x = [TT]

19.2 Theorems

[getInitMove_def]
⊢ (getInitMove [] = [NONE]) ∧
(∀xs.
    getInitMove
    (Name PlatoonSergeant says
        prop (SOME (SLc (PSG initiateMovement)))::xs) =
        [SOME (SLc (PSG initiateMovement))]) ∧
(∀xs, v2, v3. getInitMove (TT::xs) = getInitMove xs) ∧
(∀xs, v2, v3. getInitMove (FF::xs) = getInitMove xs) ∧
(∀xs, v3. getInitMove (notf v3::xs) = getInitMove xs) ∧
(∀xs, v4, v5. getInitMove (v4 andf v5::xs) = getInitMove xs) ∧
(∀xs, v7, v8. getInitMove (v0 orf v7::xs) = getInitMove xs) ∧
(∀xs, v9, v8. getInitMove (v8 impf v9::xs) = getInitMove xs) ∧
(∀xs, v10. getInitMove (v0 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v13. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v14. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v14. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v14. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v14. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says TT::xs) = getInitMove xs) ∧
(∀xs, v12. getInitMove (v12 says FF::xs) = getInitMove xs) ∧
(∀xs, v14. getInitMove (Name v13 says prop NONE::xs) =
    getInitMove xs) ∧
(∀xs, v14. getInitMove (Name PlatoonLeader says prop (SOME v14)::xs) =
    getInitMove xs) ∧
(∀xs, v15. getInitMove (v10 eqf v11::xs) = getInitMove xs)
getInitMove

(Name PlatoonSergeant says prop (SOME (ESCc v146))::
 \( xs \)) =
 getInitMove \( xs \) \( \land \)
(\( \forall \) \( xs \) \( v150 \).
 getInitMove

(Name PlatoonSergeant says prop (SOME (SLc (PL v150))))::
 \( xs \)) =
 getInitMove \( xs \) \( \land \)
(\( \forall \) \( xs \).
 getInitMove

(Name PlatoonSergeant says prop (SOME (SLc (PSG psgIncomplete)))::\( xs \) =
 getInitMove \( xs \) \( \land \)
(\( \forall \) \( xs \).
 getInitMove

(Name PlatoonSergeant says prop (SOME (SLc (PSG invalidPsgCommand)))::\( xs \) =
 getInitMove \( xs \) \( \land \)
(\( \forall \) \( xs \) \( v68 \) \( v136 \) \( v135 \).
 getInitMove (v135 meet v136 says prop v68::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v68 \) \( v138 \) \( v137 \).
 getInitMove (v137 quoting v138 says prop v68::\( xs \) =
 getInitMove \( xs \) \( \land \)
(\( \forall \) \( xs \) \( v69 \) \( v12 \).
 getInitMove (v12 says notf v69::\( xs \) = getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v71 \) \( v70 \) \( v12 \).
 getInitMove (v12 says (v70 andf v71)::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v73 \) \( v72 \) \( v12 \).
 getInitMove (v12 says (v72 orf v73)::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v75 \) \( v74 \) \( v12 \).
 getInitMove (v12 says (v74 impf v75)::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v77 \) \( v76 \) \( v12 \).
 getInitMove (v12 says (v76 eqf v77)::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v79 \) \( v78 \) \( v12 \).
 getInitMove (v12 says v78 says v79::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v81 \) \( v80 \) \( v12 \).
 getInitMove (v12 says v80 speaks_for v81::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v83 \) \( v82 \) \( v12 \).
 getInitMove (v12 says v82 controls v83::\( xs \) =
 getInitMove \( xs \)) \( \land \)
(\( \forall \) \( xs \) \( v86 \) \( v85 \) \( v84 \) \( v12 \).
getInitMove (v12 says reps v84 v85 v86::xs) =
getInitMove xs) ∧
∀ zs v88 v87 v12.
getInitMove (v12 says v87 domi v88::xs) =
getInitMove xs) ∧
∀ zs v89 v89 v12.
getInitMove (v12 says v89 eqi v90::xs) = getInitMove xs) ∧
∀ zs v92 v91 v12.
getInitMove (v12 says v91 doms v92::xs) =
getInitMove xs) ∧
∀ zs v94 v93 v12.
getInitMove (v12 says v93 eqs v94::xs) = getInitMove xs) ∧
∀ zs v96 v95 v12.
getInitMove (v12 says v95 eqn v96::xs) = getInitMove xs) ∧
∀ zs v98 v97 v12.
getInitMove (v12 says v97 lte v98::xs) = getInitMove xs) ∧
∀ zs v99 v12 v100.
getInitMove (v12 says v90 lt v100::xs) = getInitMove xs) ∧
∀ zs v15 v14.
getInitMove (v14 speaks_for v15::xs) = getInitMove xs) ∧
∀ zs v17 v16.
getInitMove (v16 controls v17::xs) = getInitMove xs) ∧
∀ zs v20 v19 v18.
getInitMove (reps v18 v19 v20::xs) = getInitMove xs) ∧
∀ zs v22 v21.
getInitMove (v21 domi v22::xs) = getInitMove xs) ∧
∀ zs v24 v23.
getInitMove (v23 eqi v24::xs) = getInitMove xs) ∧
∀ zs v26 v25.
getInitMove (v25 doms v26::xs) = getInitMove xs) ∧
∀ zs v28 v27.
getInitMove (v27 eqs v28::xs) = getInitMove xs) ∧
∀ zs v30 v29.
getInitMove (v29 eqn v30::xs) = getInitMove xs) ∧
∀ zs v32 v31.
getInitMove (v31 lte v32::xs) = getInitMove xs) ∧
∀ xs v34 v33.
getInitMove (v33 lte v34::xs) = getInitMove xs

[getInitMove_ind]

⊢ ∀ P.
  P [] ∧
  ∀ zs.
    P
    (Name PlatoonSergeant says
     prop (SOME (SLc (PSG initiateMovement))::xs)) ∧
    (∀ zs. P xs ⇒ P (TT::xs)) ∧ (∀ zs. P xs ⇒ P (FF::xs)) ∧
    (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
(∀ v₀ v₁ xs. P xs ⇒ P (v₀ orf v₁::xs)) ∧
(∀ v₈ v₀ xs. P xs ⇒ P (v₈ impf v₀::xs)) ∧
(∀ v₁₀ v₁₁ xs. P xs ⇒ P (v₁₀ eqf v₁₁::xs)) ∧
(∀ v₁₂ xs. P xs ⇒ P (v₁₂ says TT::xs)) ∧
(∀ v₁₂ xs. P xs ⇒ P (v₁₂ says FF::xs)) ∧
(∀ v₁₄ xs. P xs ⇒ P (Name v₁₄ says prop NONE::xs)) ∧
(∀ v₁₄ xs.
  P xs ⇒
  P (Name PlatoonLeader says prop (SOME v₁₄)::xs)) ∧
(∀ v₁₆ xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME (ESC v₁₆))::xs)) ∧
(∀ v₁₅₀ xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME (SLc (PL v₁₅₀)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME (SLc (PSG psgIncomplete)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME (SLc (PSG invalidPsgCommand)))::xs)) ∧
(∀ v₁₃₅ v₁₃₆ v₈₈ xs.
  P xs ⇒ P (v₁₃₅ meet v₁₃₆ says prop v₈₈::xs)) ∧
(∀ v₁₃₇ v₁₃₈ v₈₈ xs.
  P xs ⇒ P (v₁₃₇ quoting v₁₃₈ says prop v₈₈::xs)) ∧
(∀ v₁₂ v₉₉ xs. P xs ⇒ P (v₁₂ says notf v₉₉::xs)) ∧
(∀ v₁₂ v₇₀ v₇₁ xs. P xs ⇒ P (v₁₂ says (v₇₀ andf v₇₁)::xs)) ∧
(∀ v₁₂ v₇₂ v₇₃ xs. P xs ⇒ P (v₁₂ says (v₇₂ orf v₇₃)::xs)) ∧
(∀ v₁₂ v₇₄ v₇₅ xs. P xs ⇒ P (v₁₂ says (v₇₄ impf v₇₅)::xs)) ∧
(∀ v₁₂ v₇₆ v₇₇ xs. P xs ⇒ P (v₁₂ says (v₇₆ eqf v₇₇)::xs)) ∧
(∀ v₁₂ v₇₈ v₇₉ xs. P xs ⇒ P (v₁₂ says v₇₈ says v₇₉::xs)) ∧
(∀ v₁₂ v₈₀ v₈₁ xs.
  P xs ⇒ P (v₁₂ says v₈₀ speaks_for v₈₁::xs)) ∧
(∀ v₁₂ v₈₂ v₈₃ xs.
  P xs ⇒ P (v₁₂ says v₈₂ controls v₈₃::xs)) ∧
(∀ v₁₂ v₈₄ v₈₅ v₈₆ xs.
  P xs ⇒ P (v₁₂ says reps v₈₄ v₈₅ v₈₆::xs)) ∧
(∀ v₁₂ v₈₇ v₈₈ xs. P xs ⇒ P (v₁₂ says v₈₇ domi v₈₈::xs)) ∧
(∀ v₁₂ v₈₉ v₉₀ xs. P xs ⇒ P (v₁₂ says v₈₉ eqi v₉₀::xs)) ∧
(∀ v₁₂ v₉₁ v₉₂ xs. P xs ⇒ P (v₁₂ says v₉₁ doms v₉₂::xs)) ∧
(∀ v₁₂ v₉₃ v₉₄ xs. P xs ⇒ P (v₁₂ says v₉₃ eqs v₉₄::xs)) ∧
\[
(\forall v_{12} v_{95} v_{96} \cdot P \cdot x = P \quad (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::x)) \land \\
(\forall v_{12} v_{97} v_{98} \cdot P \cdot x = P \quad (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::x)) \land \\
(\forall v_{12} v_{99} v_{100} \cdot x = P \quad (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::x)) \land \\
(\forall v_{14} v_{15} \cdot P \cdot x = P \quad (v_{14} \text{ speaks for } v_{15}::x)) \land \\
(\forall v_{16} v_{17} \cdot P \cdot x = P \quad (v_{16} \text{ controls } v_{17}::x)) \land \\
(\forall v_{18} v_{19} v_{20} \cdot P \cdot x = P \quad (\text{reps } v_{18} v_{19} v_{20}::x)) \land \\
(\forall v_{21} v_{22} \cdot P \cdot x = P \quad (v_{21} \text{ domi } v_{22}::x)) \land \\
(\forall v_{23} v_{24} \cdot P \cdot x = P \quad (v_{23} \text{ eqi } v_{24}::x)) \land \\
(\forall v_{25} v_{26} \cdot P \cdot x = P \quad (v_{25} \text{ doms } v_{26}::x)) \land \\
(\forall v_{27} v_{28} \cdot P \cdot x = P \quad (v_{27} \text{ eqs } v_{28}::x)) \land \\
(\forall v_{29} v_{30} \cdot P \cdot x = P \quad (v_{29} \text{ eqn } v_{30}::x)) \land \\
(\forall v_{31} v_{32} \cdot P \cdot x = P \quad (v_{31} \text{ lte } v_{32}::x)) \land \\
(\forall v_{33} v_{34} \cdot P \cdot x = P \quad (v_{33} \text{ lt } v_{34}::x)) \land \\
\forall v. P \cdot v
\]

\[\text{getPlCom_def}\]

\[
\vdash (\text{getPlCom } [] = \text{invalidPlCommand}) \land \\
(\forall x. \text{cmd}. \\
\quad \text{getPlCom} \\
\quad \quad (\text{Name PlatoonLeader says prop } (\text{SOME } (\text{SLc } (\text{PL cmd}))::x)) = \\
\quad \quad \text{cmd}) \land (\forall x. \text{getPlCom } (\text{TT::x}) = \text{getPlCom } x) \land \\
(\forall x. \text{getPlCom } (\text{FF::x}) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{prop } v_2::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{notf } v_3::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v4 andf } v_5::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v6 orf } v_7::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v8 impf } v_9::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v10 eqf } v_{11}::x) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v12 says } \text{TT::x}) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v12 says } \text{FF::x}) = \text{getPlCom } x) \land \\
(\forall v. \text{getPlCom } (\text{v134 says prop } \text{NONE::x}) = \text{getPlCom } x) \land \\
(\forall v. \text{v146}.
\quad \text{getPlCom} \\
\quad \quad (\text{Name PlatoonLeader says prop } (\text{SOME } (\text{ESCc } v_{146}))::x) = \\
\quad \quad \text{getPlCom } x) \land \\
(\forall v. \text{v151}).
\quad \text{getPlCom} \\
\quad \quad (\text{Name PlatoonLeader says prop } (\text{SOME } (\text{PSG } v_{151})))::x) = \\
\quad \quad \text{getPlCom } x) \land \\
(\forall v. \text{v144}).
\quad \text{getPlCom} \\
\quad \quad (\text{Name PlatoonSergeant says prop } (\text{SOME } v_{144})))::x) = \\
\quad \quad \text{getPlCom } x) \land \\
(\forall v. \text{v168} v_{136} v_{135}.
\quad \text{getPlCom } (v_{135} \text{ meet } v_{136} \text{ says prop } v_{168}::x) = \\
\quad \text{getPlCom } x) \land
\]
\((\forall xs\ v_{98}\ v_{138}\ v_{137}).\)

\(\text{getPlCom}\ (v_{137}\ \text{quoting}\ v_{138}\ \text{says prop}\ v_{98}::xs) = \\text{getPlCom}\ xs\) \land

\((\forall xs\ v_{99}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says notf}\ v_{99}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{71}\ v_{70}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ (v_{70}\ \text{andf}\ v_{71})::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{73}\ v_{72}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ (v_{72}\ \text{orf}\ v_{73})::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{75}\ v_{74}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ (v_{74}\ \text{impf}\ v_{75})::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{77}\ v_{76}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ (v_{76}\ \text{eqf}\ v_{77})::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{79}\ v_{78}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{78}\ \text{says}\ v_{79}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{81}\ v_{80}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{80}\ \text{speaks for}\ v_{81}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{83}\ v_{82}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{82}\ \text{controls}\ v_{83}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{86}\ v_{85}\ v_{84}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{84}\ v_{85}\ v_{86}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{88}\ v_{87}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{87}\ \text{domi}\ v_{88}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{90}\ v_{90}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{90}\ \text{eqi}\ v_{90}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{92}\ v_{91}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{91}\ \text{doms}\ v_{92}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{94}\ v_{93}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{93}\ \text{eqs}\ v_{94}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{96}\ v_{95}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{95}\ \text{eqn}\ v_{96}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{98}\ v_{97}\ v_{12}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{97}\ \text{lte}\ v_{98}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{99}\ v_{12}\ v_{100}).\)

\(\text{getPlCom}\ (v_{12}\ \text{says}\ v_{99}\ \text{lt}\ v_{100}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{15}\ v_{14}).\)

\(\text{getPlCom}\ (v_{14}\ \text{speaks for}\ v_{15}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{17}\ v_{16}).\)

\(\text{getPlCom}\ (v_{16}\ \text{controls}\ v_{17}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{20}\ v_{19}\ v_{18}).\)

\(\text{getPlCom}\ (\text{reps}\ v_{18}\ v_{19}\ v_{20}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{22}\ v_{21}).\)

\(\text{getPlCom}\ (v_{21}\ \text{domi}\ v_{22}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{24}\ v_{23}).\)

\(\text{getPlCom}\ (v_{23}\ \text{eqi}\ v_{24}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{26}\ v_{25}).\)

\(\text{getPlCom}\ (v_{25}\ \text{doms}\ v_{26}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{28}\ v_{27}).\)

\(\text{getPlCom}\ (v_{27}\ \text{eqs}\ v_{28}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{30}\ v_{29}).\)

\(\text{getPlCom}\ (v_{29}\ \text{eqn}\ v_{30}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{32}\ v_{31}).\)

\(\text{getPlCom}\ (v_{31}\ \text{lte}\ v_{32}::xs) = \text{getPlCom}\ xs\) \land

\((\forall xs\ v_{34}\ v_{33}).\)

\(\text{getPlCom}\ (v_{33}\ \text{lt}\ v_{34}::xs) = \text{getPlCom}\ xs\) \land

\(88\)
\[
\text{getPlCom\_ind}
\]
\[
\vdash \forall P .
\]
\[
P \; [ ] \land
\]
\[
(\forall \text{cmd} \; x) .
\]
\[
P
\]
\[
(\text{Name PlatoonLeader says prop (SOME (S\text{Lc (PL cmd))}::} \; x)) \land (\forall x \; P \; x) \Rightarrow P \; (\text{TT}::x) \land
\]
\[
(\forall x \; P \; x) \Rightarrow P \; (\text{FF}::x) \land
\]
\[
(\forall v2 \; x) .
\]
\[
P \; x \Rightarrow P \; (\text{prop v2::x}) \land
\]
\[
(\forall v3 \; x) .
\]
\[
P \; x \Rightarrow P \; (\text{notf v3::x}) \land
\]
\[
(\forall v5 v8 \; x) .
\]
\[
P \; x \Rightarrow P \; (v4 \; \text{andf v5::x}) \land
\]
\[
(\forall v6 v7 v8 \; x) .
\]
\[
P \; x \Rightarrow P \; (v6 \; \text{orf v7::x}) \land
\]
\[
(\forall v8 v9 \; x) .
\]
\[
P \; x \Rightarrow P \; (v8 \; \text{impf v9::x}) \land
\]
\[
(\forall v10 v11 v12 v134 v146 \; x) .
\]
\[
P \; x \Rightarrow P \; (\text{Name v134 says prop NONE::x}) \land
\]
\[
(\forall v134 \; x) .
\]
\[
P \; x \Rightarrow P
\]
\[
(\text{Name PlatoonLeader says prop (SOME (ES\text{Cc v146))}::} \; x)) \land
\]
\[
(\forall v151 \; x) .
\]
\[
P \; x \Rightarrow
\]
\[
(\text{Name PlatoonLeader says prop (SOME (S\text{Lc (PSG v151))}::} \; x)) \land
\]
\[
(\forall v144 \; x) .
\]
\[
P \; x \Rightarrow P
\]
\[
(\text{Name PlatoonSergeant says prop (SOME v144)::x}) \land
\]
\[
(\forall v135 v136 v88 \; x) .
\]
\[
P \; x \Rightarrow P \; (v135 \; \text{meet v136 says prop v88::x}) \land
\]
\[
(\forall v137 v138 v88 \; x) .
\]
\[
P \; x \Rightarrow P \; (v137 \; \text{quoting v138 says prop v88::x}) \land
\]
\[
(\forall v12 v69 v89 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says notf v90::x}) \land
\]
\[
(\forall v12 v70 v71 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says (v70 \; andf v71::x)}) \land
\]
\[
(\forall v12 v72 v73 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says (v72 \; orf v73::x)}) \land
\]
\[
(\forall v12 v74 v75 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says (v74 \; impf v75::x)}) \land
\]
\[
(\forall v12 v76 v77 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says (v76 \; eqf v77::x)}) \land
\]
\[
(\forall v12 v78 v79 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says v78 says v79::x}) \land
\]
\[
(\forall v12 v80 v81 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says v80 speaks for v81::x}) \land
\]
\[
(\forall v12 v82 v83 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{says v82 controls v83::x}) \land
\]
\[
(\forall v12 v84 v85 v86 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{saye reps v84 v85 v86::x}) \land
\]
\[
(\forall v12 v87 v88 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{saye v87 domi v88::x}) \land
\]
\[
(\forall v12 v89 v90 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{saye v89 eqi v90::x}) \land
\]
\[
(\forall v12 v91 v92 \; x) .
\]
\[
P \; x \Rightarrow P \; (v12 \; \text{saye v91 doms v92::x}) \land
\]
\[
(\forall v_{12} v_{14} x_s. \ P \ x_s \Rightarrow P (v_{12} \text{ says } v_{14} \text{ eqs } v_{14} :: x_s)) \land \\
(\forall v_{12} v_{15} v_{26} x_s. \ P \ x_s \Rightarrow P (v_{12} \text{ says } v_{15} \text{ eqn } v_{26} :: x_s)) \land \\
(\forall v_{12} v_{27} v_{34} x_s. \ P \ x_s \Rightarrow P (v_{12} \text{ says } v_{27} \text{ lte } v_{34} :: x_s)) \land \\
(\forall v_{12} v_{99} v_{100} x_s. \ P \ x_s \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100} :: x_s)) \land \\
(\forall v_{14} v_{15} x_s. \ P \ x_s \Rightarrow P (v_{14} \text{ speaks_for } v_{15} :: x_s)) \land \\
(\forall v_{16} v_{17} x_s. \ P \ x_s \Rightarrow P (v_{16} \text{ controls } v_{17} :: x_s)) \land \\
(\forall v_{18} v_{19} v_{20} x_s. \ P \ x_s \Rightarrow P (\text{reps } v_{18} v_{19} v_{20} :: x_s)) \land \\
(\forall v_{21} v_{22} x_s. \ P \ x_s \Rightarrow P (v_{21} \text{ domi } v_{22} :: x_s)) \land \\
(\forall v_{22} v_{24} x_s. \ P \ x_s \Rightarrow P (v_{22} \text{ eqi } v_{24} :: x_s)) \land \\
(\forall v_{25} v_{26} x_s. \ P \ x_s \Rightarrow P (v_{25} \text{ doms } v_{26} :: x_s)) \land \\
(\forall v_{27} v_{28} x_s. \ P \ x_s \Rightarrow P (v_{27} \text{ eqs } v_{28} :: x_s)) \land \\
(\forall v_{29} v_{30} x_s. \ P \ x_s \Rightarrow P (v_{29} \text{ eqn } v_{30} :: x_s)) \land \\
(\forall v_{31} v_{32} x_s. \ P \ x_s \Rightarrow P (v_{31} \text{ lte } v_{32} :: x_s)) \land \\
(\forall v_{33} v_{34} x_s. \ P \ x_s \Rightarrow P (v_{33} \text{ lt } v_{34} :: x_s)) \Rightarrow \\
\forall v. \ P \ v
\]

\[\text{getPsgCom_def}\]

\[\forall x. \ \text{cmd}. \ \\
\text{getPsgCom} \ \\
\quad \ (\text{Name PlatoonSergeant says prop } (\text{SOME} \ (\text{SLc} \ (\text{PSG cmd}))) :: x_s) = \ \\
\quad \ cmd) \land (\forall x. \ \text{getPsgCom} \ (\text{TT} :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_2. \ \text{getPsgCom} \ (\text{prop} \ v_2 :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_3. \ \text{getPsgCom} \ (\text{notif} \ v_3 :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_4. \ \text{getPsgCom} \ (v_4 \text{ andf } v_5 :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_6. \ \text{getPsgCom} \ (v_6 \text{ orf } v_7 :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_8. \ \text{getPsgCom} \ (v_8 \text{ impf } v_9 :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{10}. \ \text{getPsgCom} \ (v_{10} \text{ eqf } v_{11} :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{11}. \ \text{getPsgCom} \ (v_{12} \text{ says } \text{TT} :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{12}. \ \text{getPsgCom} \ (v_{12} \text{ says } \text{FF} :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{134}. \ \\
\quad \ \text{getPsgCom} \ (\text{Name } v_{134} \text{ says prop } \text{NONE} :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{144}. \ \\
\quad \ \text{getPsgCom} \ (\text{Name } \text{PlatoonLeader says prop } (\text{SOME} \ v_{144}) :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{146}. \ \\
\quad \ \text{getPsgCom} \ (\text{Name } \text{PlatoonSergeant says prop } (\text{SOME} \ v_{146}) :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{150}. \ \\
\quad \ \text{getPsgCom} \ (\text{Name } \text{PlatoonSergeant says prop } (\text{SOME} \ v_{150}) :: x_s) = \text{getPsgCom} \ x_s) \land \\
\quad \ (\forall x. \ v_{98} v_{136} v_{135}. \ \\
\quad \ \text{getPsgCom} (v_{136} \text{ meet } v_{136} \text{ says prop } v_{98} :: x_s) = \]
getPsgCom (xs) \land
\forall xs \; v_{08} \; v_{138} \; v_{137}.
getPsgCom (v_{137} \text{ quoting } v_{138} \text{ says prop } v_{09} :: xs) =
getPsgCom (xs) \land
\forall xs \; v_{09} \; v_{12}.
getPsgCom (v_{12} \text{ says notf } v_{09} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{17} \; v_{170} \; v_{172}.
getPsgCom (v_{17} \text{ speaks_for } v_{170} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{73} \; v_{72} \; v_{172}.
getPsgCom (v_{72} \text{ says } (\text{v}_7 \text{ domf } v_{73}) :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{75} \; v_{74} \; v_{172}.
getPsgCom (v_{74} \text{ says } (\text{v}_7 \text{ orf } v_{75}) :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{77} \; v_{76} \; v_{172}.
getPsgCom (v_{76} \text{ eqf } v_{77}) :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{79} \; v_{78} \; v_{172}.
getPsgCom (v_{78} \text{ says } v_{79} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{81} \; v_{80} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{80} \text{ speaks_for } v_{81} :: xs) =
getPsgCom (xs) \land
\forall xs \; v_{83} \; v_{82} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{82} \text{ controls } v_{83} :: xs) =
getPsgCom (xs) \land
\forall xs \; v_{86} \; v_{85} \; v_{84} \; v_{172}.
getPsgCom (v_{12} \text{ says } \text{reps } v_{84} \; v_{85} \; v_{86} :: xs) =
getPsgCom (xs) \land
\forall xs \; v_{88} \; v_{87} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{87} \text{ domi } v_{88} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{90} \; v_{89} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{89} \text{ eqi } v_{90} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{92} \; v_{91} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{91} \text{ doms } v_{92} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{94} \; v_{93} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{93} \text{ eqs } v_{94} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{96} \; v_{95} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{95} \text{ eqn } v_{96} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{98} \; v_{97} \; v_{172}.
getPsgCom (v_{12} \text{ says } v_{97} \text{ lte } v_{98} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{99} \; v_{12} \; v_{100}.
getPsgCom (v_{12} \text{ says } v_{99} \text{ lt } v_{100} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{15} \; v_{14}.
getPsgCom (v_{14} \text{ speaks_for } v_{15} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{17} \; v_{16}.
getPsgCom (v_{16} \text{ controls } v_{17} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{20} \; v_{19} \; v_{18}.
getPsgCom (\text{reps } v_{18} \; v_{19} \; v_{20} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{22} \; v_{21}.
getPsgCom (v_{21} \text{ domi } v_{22} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{24} \; v_{23}.
getPsgCom (v_{23} \text{ eqi } v_{24} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{26} \; v_{25}.
getPsgCom (v_{25} \text{ doms } v_{26} :: xs) = \text{getPsgCom } (xs) \land
\forall xs \; v_{28} \; v_{27}.
getPsgCom (v_{27} \text{ eqs } v_{28} :: xs) = \text{getPsgCom } (xs) \land

91
\[(\forall xs \; v_{10} \; v_{20} . \; \text{getPsgCom}(v_{20} \; \text{eqn} \; v_{10} :: xs) = \text{getPsgCom}(xs)) \land \]
\[(\forall xs \; v_{12} \; v_{21} . \; \text{getPsgCom}(v_{21} \; \text{lte} \; v_{12} :: xs) = \text{getPsgCom}(xs)) \land \]
\[(\forall xs \; v_{34} \; v_{33} . \; \text{getPsgCom}(v_{33} \; \text{lt} \; v_{34} :: xs) = \text{getPsgCom}(xs)) \land \]

\[\vdash \forall P . \]
\[P [] \land \]
\[(\forall cmd \; xs . \]
\[P \]
\[\quad (\text{Name PlatoonSergeant says} \]
\[\quad \quad \text{prop(\text{SOME (SLc (\text{PSG cmd}) :: xs)})} \land \]
\[\quad (\forall xs . \; P \; xs \Rightarrow P(\text{TT} :: xs)) \land (\forall xs . \; P \; xs \Rightarrow P(\text{FF} :: xs)) \land \]
\[\quad (\forall v_2 \; xs . \; P \; xs \Rightarrow P(\text{prop} \; v_2 :: xs)) \land \]
\[\quad (\forall v_3 \; xs . \; P \; xs \Rightarrow P(\text{notf} \; v_3 :: xs)) \land \]
\[\quad (\forall v_4 \; v_5 \; xs . \; P \; xs \Rightarrow P(\text{v_4 andf v_5 :: xs})) \land \]
\[\quad (\forall v_6 \; v_7 \; xs . \; P \; xs \Rightarrow P(\text{v_6 orf v_7 :: xs})) \land \]
\[\quad (\forall v_8 \; v_9 \; xs . \; P \; xs \Rightarrow P(\text{v_8 impf v_9 :: xs})) \land \]
\[\quad (\forall v_{10} \; v_{11} \; xs . \; P \; xs \Rightarrow P(\text{v_{10 eqf v_{11} :: xs}})) \land \]
\[\quad (\forall v_{12} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says TT :: xs}})) \land \]
\[\quad (\forall v_{12} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says FF :: xs}})) \land \]
\[\quad (\forall v_{134} \; xs . \; P \; xs \Rightarrow P(\text{Name v_{134} says prop NONE :: xs})) \land \]
\[\quad (\forall v_{144} \; xs . \]
\[\quad \quad P \; xs \Rightarrow \]
\[\quad \quad P(\text{Name PlatoonLeader says prop (\text{SOME v_{144} :: xs})}) \land \]
\[\quad (\forall v_{146} \; xs . \]
\[\quad \quad P \; xs \Rightarrow \]
\[\quad \quad P \]
\[\quad \quad (\text{Name PlatoonSergeant says prop (\text{SOME (ESLc v_{146} :: xs})}) \land \]
\[\quad (\forall v_{150} \; xs . \]
\[\quad \quad P \; xs \Rightarrow \]
\[\quad \quad P \]
\[\quad \quad (\text{Name PlatoonSergeant says prop (\text{SOME (SLc (PL v_{150}) :: xs})}) \land \]
\[\quad (\forall v_{135} \; v_{136} \; v_{68} \; xs . \]
\[\quad \quad P \; xs \Rightarrow P(\text{v_{135 meet v_{136} says prop v_{68} :: xs}}) \land \]
\[\quad (\forall v_{137} \; v_{138} \; v_{68} \; xs . \]
\[\quad \quad P \; xs \Rightarrow P(\text{v_{137 quoting v_{138} says prop v_{68} :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{69} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says notf v_{69} :: xs}})) \land \]
\[\quad (\forall v_{12} \; v_{70} \; v_{71} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says (v_{70} andf v_{71}) :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{72} \; v_{73} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says (v_{72} orf v_{73}) :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{74} \; v_{75} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says (v_{74} impf v_{75}) :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{76} \; v_{77} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says (v_{76 eqf v_{77}) :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{78} \; v_{79} \; xs . \; P \; xs \Rightarrow P(\text{v_{12 says v_{78} eqf v_{79} :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{80} \; v_{81} \; xs . \]
\[\quad \quad P \; xs \Rightarrow P(\text{v_{12 says v_{80} speaks_for v_{81} :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{82} \; v_{83} \; xs . \]
\[\quad \quad P \; xs \Rightarrow P(\text{v_{12 says v_{82} controls v_{83} :: xs}}) \land \]
\[\quad (\forall v_{12} \; v_{84} \; v_{85} \; v_{86} \; xs . \]

92
\[ P \ \text{xs} \Rightarrow P (v_{12} \ \text{says reps} \ \text{v84 v95 v66::xs}) \land \\
(\forall v_{12} v_{97} v_{98} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v97 domi v98::xs}) \land \\
(\forall v_{12} v_{99} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v99 eqi v00::xs}) \land \\
(\forall v_{12} v_{91} v_{92} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v91 doms v92::xs}) \land \\
(\forall v_{12} v_{93} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v93 eqs v94::xs}) \land \\
(\forall v_{12} v_{95} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v95 eqn v96::xs}) \land \\
(\forall v_{12} v_{97} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v97 lte v98::xs}) \land \\
(\forall v_{12} v_{99} v_{100} x_{12} . \ P \ x_{12} \Rightarrow P (v_{12} \ \text{says v99 lt v100::xs}) \land \\
(\forall v_{14} v_{15} x_{12} . \ P \ x_{12} \Rightarrow P (v_{14} \ \text{speaks_for v15::xs}) \land \\
(\forall v_{16} v_{17} x_{12} . \ P \ x_{12} \Rightarrow P (v_{16} \ \text{controls v17::xs}) \land \\
(\forall v_{18} v_{19} v_{20} x_{12} . \ P \ x_{12} \Rightarrow P (\ \text{reps v18 v19 v20::xs}) \land \\
(\forall v_{21} v_{22} x_{12} . \ P \ x_{12} \Rightarrow P (v_{21} \ \text{domi v22::xs}) \land \\
(\forall v_{23} v_{24} x_{12} . \ P \ x_{12} \Rightarrow P (v_{23} \ \text{eqi v24::xs}) \land \\
(\forall v_{25} v_{26} x_{12} . \ P \ x_{12} \Rightarrow P (v_{25} \ \text{doms v26::xs}) \land \\
(\forall v_{27} v_{28} x_{12} . \ P \ x_{12} \Rightarrow P (v_{27} \ \text{eqs v28::xs}) \land \\
(\forall v_{29} v_{30} x_{12} . \ P \ x_{12} \Rightarrow P (v_{29} \ \text{eqn v30::xs}) \land \\
(\forall v_{31} v_{32} x_{12} . \ P \ x_{12} \Rightarrow P (v_{31} \ \text{lte v32::xs}) \land \\
(\forall v_{33} v_{34} x_{12} . \ P \ x_{12} \Rightarrow P (v_{33} \ \text{lt v34::xs}) \Rightarrow \\
\forall v. \ P \ v \\
\text{getRecon_def} \]

\[ \vdash (\text{getRecon [ ]} = [\text{NONE}]) \land \\
\forall x_{12} . \\
\quad \text{getRecon} \\
\quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))::} \\
\quad \text{xs}) = \\
\quad [\text{SOME (SLc (PL recon))}] \land \\
\quad (\forall x_{12} . \ \text{getRecon (TT::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} . \ \text{getRecon (FF::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (prop v12::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (notf v12::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (andf v12::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (orf v12::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (impf v12::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{10} . \ \text{getRecon (v10 eqf v11::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (v12 says TT::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} v_{12} . \ \text{getRecon (v12 says FF::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} \ \text{v134} . \\
\quad \text{getRecon (Name v134 says prop NONE::xs)} = \text{getRecon xs}) \land \\
\quad (\forall x_{12} \ \text{v146} . \\
\quad \text{getRecon} \\
\quad (\text{Name PlatoonLeader says prop (SOME (EScc v146)::xs)} = \\
\quad \text{getRecon xs}) \land \\
\quad (\forall x_{12} . \\
\quad \text{getRecon} \\
\quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL receiveMission))::xs)} = \\
\quad \text{getRecon xs}) \land \\
\quad (\forall x_{12} .} \]
PLANPBDEF THEORY

Theorems

getRecon
   (Name PlatoonLeader says prop (SOME (SLc (PL warno)))::
     xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL tentativePlan)))::xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL report1)))::xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL report2)))::xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL complete)))::xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL plIncomplete)))::xs) =
   getRecon xs) ∧
(∀ xs.
   getRecon
   (Name PlatoonLeader says
     prop (SOME (SLc (PL invalidPlCommand)))::xs) =
   getRecon xs) ∧
\( \forall x s \ v151. \)
getRecon
  (Name PlatoonLeader says prop (SOME (SLc (PSG v151))::xs) =
getRecon xs) \land
(\forall x s \ v144.
getRecon
  (Name PlatoonSergeant says prop (SOME v144::xs) =
getRecon xs) \land
(\forall x s \ v138 \ v136 \ v135.
getRecon (v135 meet v136 says prop \( v_{08} :: xs \)) =
getRecon xs) \land
(\forall x s \ v137.
getRecon (v137 quoting v138 says prop \( v_{08} :: xs \)) =
getRecon xs) \land
(\forall x s \ v_{09} \ v_{12}.
getRecon (v_{12} says notf \( v_{09} :: xs \)) =
getRecon xs) \land
(\forall x s \ v_{71} \ v_{70} \ v_{12}.
getRecon (v_{12} says (v_{70} andf v_{71})::xs) =
getRecon xs) \land
(\forall x s \ v_{73} \ v_{72} \ v_{12}.
getRecon (v_{12} says (v_{72} orf v_{73})::xs) =
getRecon xs) \land
(\forall x s \ v_{75} \ v_{74} \ v_{12}.
getRecon (v_{12} says (v_{74} impf v_{75})::xs) =
getRecon xs) \land
(\forall x s \ v_{77} \ v_{76} \ v_{12}.
getRecon (v_{12} says (v_{76} eqf v_{77})::xs) =
getRecon xs) \land
(\forall x s \ v_{79} \ v_{78} \ v_{12}.
getRecon (v_{12} says v_{78} says v_{79}::xs) =
getRecon xs) \land
(\forall x s \ v_{81} \ v_{80} \ v_{12}.
getRecon (v_{12} says v_{80} speaks_for v_{81}::xs) =
getRecon xs) \land
(\forall x s \ v_{83} \ v_{82} \ v_{12}.
getRecon (v_{12} says v_{82} controls v_{83}::xs) =
getRecon xs) \land
(\forall x s \ v_{86} \ v_{85} \ v_{84} \ v_{12}.
getRecon (v_{12} says reps v_{84} v_{85} v_{86}::xs) =
getRecon xs) \land
(\forall x s \ v_{88} \ v_{87} \ v_{12}.
getRecon (v_{12} says v_{87} domi v_{88}::xs) =
getRecon xs) \land
(\forall x s \ v_{90} \ v_{89} \ v_{12}.
getRecon (v_{12} says v_{89} eqi v_{90}::xs) =
getRecon xs) \land
(\forall x s \ v_{92} \ v_{91} \ v_{12}.
getRecon (v_{12} says v_{91} doms v_{92}::xs) =
getRecon xs) \land
(\forall x s \ v_{94} \ v_{93} \ v_{12}.
getRecon (v_{12} says v_{93} eqs v_{94}::xs) =
getRecon xs) \land
(\forall x s \ v_{96} \ v_{95} \ v_{12}.
getRecon (v_{12} says v_{95} eqn v_{96}::xs) =
getRecon xs) \land
(\forall x s \ v_{98} \ v_{97} \ v_{12}.
getRecon (v_{12} says v_{97} lte v_{98}::xs) =
getRecon xs) \land
(\forall x s \ v_{99} \ v_{12} \ v100.
getRecon (v_{12} says v_{99} lt v100::xs) =
getRecon xs) \land
(\forall x s \ v_{15} \ v_{14}.

getRecon \(v_{14} \text{ speaks for } v_{15}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{17} \quad v_{10} \quad v_0\).
getRecon \(v_{16} \text{ controls } v_{17}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{20} \quad v_{10} \quad v_{18}\).
getRecon \(\text{reps } v_{18} \quad v_{10} \quad v_{20}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{22} \quad v_{21} \quad v_0\).
getRecon \(v_{21} \quad \text{doms } v_{22}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{24} \quad v_{23} \quad v_0\).
getRecon \(v_{23} \quad \text{eqi } v_{24}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{26} \quad v_{25} \quad v_0\).
getRecon \(v_{25} \quad \text{doms } v_{26}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{28} \quad v_{27} \quad v_0\).
getRecon \(v_{27} \quad \text{eqs } v_{28}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{30} \quad v_{29} \quad v_0\).
getRecon \(v_{29} \quad \text{eqn } v_{30}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{32} \quad v_{31} \quad v_0\).
getRecon \(v_{31} \quad \text{lte } v_{32}::xs\) = getRecon \(xs\) \land
\(\forall xs \quad v_{34} \quad v_{33} \quad v_0\).
getRecon \(v_{33} \quad \text{lt } v_{34}::xs\) = getRecon \(xs\)

[getRecon\_ind] 
\[ \vdash \forall P. \]
\[ P \quad \land \]
\[ (\forall xs. \quad P) \]
\[ \left(\begin{align*}
(\text{Name PlatoonLeader says } \\
(\text{prop } (\text{SOME (SLc (PL recon)))::xs))) \land
(\forall xs. \quad P \quad \Rightarrow \quad P (TT::xs)) \land \quad (\forall xs. \quad P \quad \Rightarrow \quad P (FF::xs)) \land \\
(\forall v_2 \quad xs. \quad P \quad \Rightarrow \quad P (\text{prop } v_2::xs)) \land \\
(\forall v_3 \quad xs. \quad P \quad \Rightarrow \quad P (\text{notf } v_3::xs)) \land \\
(\forall v_4 \quad v_5 \quad xs. \quad P \quad \Rightarrow \quad P (v_4 \quad \text{andf } v_5::xs)) \land \\
(\forall v_6 \quad v_7 \quad xs. \quad P \quad \Rightarrow \quad P (v_6 \quad \text{orf } v_7::xs)) \land \\
(\forall v_8 \quad v_9 \quad xs. \quad P \quad \Rightarrow \quad P (v_8 \quad \text{impf } v_9::xs)) \land \\
(\forall v_{10} \quad v_11 \quad xs. \quad P \quad \Rightarrow \quad P (v_{10} \quad \text{eqf } v_{11}::xs)) \land \\
(\forall v_{12} \quad xs. \quad P \quad \Rightarrow \quad P (v_{12} \quad \text{says } TT::xs)) \land \\
(\forall v_{12} \quad xs. \quad P \quad \Rightarrow \quad P (v_{12} \quad \text{says } FF::xs)) \land \\
(\forall v_{13} \quad xs. \quad P \quad \Rightarrow \quad P (\text{Name } v_{13} \quad \text{says } \text{prop NONE::xs})) \land \\
(\forall v_{14} \quad xs. \quad P \quad \Rightarrow \quad P (\text{Name } v_{14} \quad \text{says } \text{prop NONE::xs})) \land
\end{align*}\right)
\]
\[ P \quad \Rightarrow \]
\[ P \\
\left(\begin{align*}
(\text{Name PlatoonLeader says } \\
(\text{prop } (\text{SOME (ESCc } v_{14}))))::xs))) \land
(\forall xs. \quad P) \quad \Rightarrow \quad P \\
(\text{Name PlatoonLeader says } \\
(\text{prop } (\text{SOME (SLc (PL receiveMission)))::xs})) \land \\
(\forall xs. \quad P) \quad \Rightarrow \quad P \\
(\text{Name PlatoonLeader says } \\
(\text{prop } (\text{SOME (SLc (PL warno)))::xs})) \land \\
(\forall xs. \quad P) \quad \Rightarrow \quad P \\
(\text{Name PlatoonLeader says }
\right)
\[\forall x_s. (\text{prop (SOME (SLc (PL tentativePlan)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL report1)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL completePlan)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL opioid)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL supervise)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL report2)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL complete)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL plIncomplete)))::xs}) \land \]
\[P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL invalidPlCommand)))::xs}) \land \]
\[\forall s\{\text{v151}\}. P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PSG v151)))::xs}) \land \]
\[\forall s\{\text{v144}\}. P x_s \Rightarrow P\]
\[(\text{Name PlatoonSergeant says prop (SOME v144)::xs}) \land \]
\[\text{prop (SOME (SOMAX (PL tentativePlan))))::xs}) \land \]
\[\forall x_s. P x_s \Rightarrow P\]
\[(\text{Name PlatoonLeader says prop (SOME (SLc (PL supervis}))::xs}) \land \]
(∀ v135 v136 v8 v68 xs).

\[ P \ mathit{xs} \Rightarrow P (v135 \ mathit{meet} v136 \ mathit{prop} v8\mathit{v68::xs}) \land \]
(∀ v137 v138 v8 v68 xs).

\[ P \ mathit{xs} \Rightarrow P (v137 \ mathit{quoting} v138 \ mathit{prop} v8\mathit{v68::xs}) \land \]
(∀ v12 v69 v68 v69 x). P x \Rightarrow P (v12 \ mathit{says} \mathit{notf} v8\mathit{v69::xs}) \land
(∀ v12 v70 v71 v69 v68 x). P x \Rightarrow P (v12 \ mathit{says} (v70 \ mathit{andf} v71::xs)) \land
(∀ v12 v72 v73 v69 v68 x). P x \Rightarrow P (v12 \ mathit{says} (v72 \ mathit{orf} v73::xs)) \land
(∀ v12 v74 v75 v69 v68 x). P x \Rightarrow P (v12 \ mathit{says} (v74 \ mathit{imf} v75::xs)) \land
(∀ v12 v76 v77 v69 v68 x). P x \Rightarrow P (v12 \ mathit{says} (v76 \ mathit{eqf} v77::xs)) \land
(∀ v12 v78 v79 v69 v68 x). P x \Rightarrow P (v12 \ mathit{says} v8 v78 \ mathit{says} v79::xs) \land
(∀ v12 v80 v81 v68 x).

\[ P \ mathit{xs} \Rightarrow P (v12 \ mathit{says} v80 \ mathit{speaks_for} v81::xs)) \land \]
(∀ v12 v82 v83 v68 x).

\[ P \ mathit{xs} \Rightarrow P (v12 \ mathit{says} v82 \ mathit{controls} v83::xs)) \land \]
(∀ v12 v84 v85 v86 v68 x).

\[ P \ mathit{xs} \Rightarrow P (v12 \ mathit{says} \mathit{reps} v84 v85 v86::xs)) \land \]
(∀ v12 v87 v88 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v87 \mathit{domi} v88::xs) \land
(∀ v12 v89 v86 v88 v68 x). P x \Rightarrow P (v12 \ mathit{says} v89 \mathit{eqi} v86::xs) \land
(∀ v12 v90 v91 v92 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v91 \mathit{domi} v92::xs) \land
(∀ v12 v93 v94 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v93 \mathit{eqs} v94::xs) \land
(∀ v12 v95 v96 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v95 \mathit{eqn} v96::xs) \land
(∀ v12 v97 v98 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v97 \mathit{lte} v98::xs) \land
(∀ v12 v99 v100 v68 v87 x). P x \Rightarrow P (v12 \ mathit{says} v99 \mathit{ltu} v100::xs) \land
(∀ v12 v101 v102 v68 v87 x). P x \Rightarrow P (v12 \mathit{speaks_for} v101::xs)) \land \]
(∀ v12 v103 v104 v68 v87 x). P x \Rightarrow P (v12 \mathit{controls} v103::xs)) \land
(∀ v12 v105 v106 v68 v87 x). P x \Rightarrow P (\mathit{reps} v105 v106 v87::xs)) \land
(∀ v12 v107 v108 v68 v87 x). P x \Rightarrow P (v12 \mathit{domi} v107::xs)) \land
(∀ v12 v109 v110 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqi} v109::xs)) \land
(∀ v12 v111 v112 v68 v87 x). P x \Rightarrow P (v12 \mathit{domi} v112::xs)) \land
(∀ v12 v113 v114 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqs} v113::xs)) \land
(∀ v12 v115 v116 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqn} v115::xs)) \land
(∀ v12 v117 v118 v68 v87 x). P x \Rightarrow P (v12 \mathit{lte} v117::xs)) \land
(∀ v12 v119 v120 v68 v87 x). P x \Rightarrow P (v12 \mathit{ltu} v119::xs)) \land
(∀ v12 v121 v122 v68 v87 x). P x \Rightarrow P (v12 \mathit{domi} v121::xs)) \land
(∀ v12 v123 v124 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqi} v123::xs)) \land
(∀ v12 v125 v126 v68 v87 x). P x \Rightarrow P (v12 \mathit{domi} v125::xs)) \land
(∀ v12 v127 v128 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqs} v127::xs)) \land
(∀ v12 v129 v130 v68 v87 x). P x \Rightarrow P (v12 \mathit{eqn} v129::xs)) \land
(∀ v12 v131 v132 v68 v87 x). P x \Rightarrow P (v12 \mathit{lte} v131::xs)) \land
(∀ v12 v133 v134 v68 v87 x). P x \Rightarrow P (v12 \mathit{ltu} v133::xs)) \Rightarrow
∀ v. P v

[getReport_def]

\[ \vdash \text{getReport} \mathit{[]} = \text{[NONE]} \land \]

\[ (\forall \mathit{xs}. \]

\[ \text{getReport} \]

\[ (\text{Name PlatoonLeader says} \]

\[ \text{prop} (\text{SOME} (\text{SLc (PL report1)}):\mathit{xs}) = \]

\[ \text{[SOME} (\text{SLc (PL report1)}):\mathit{xs}) \land \]

\[ (\forall \mathit{xs}. \text{getReport} (\text{TT}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs}. \text{getReport} (\text{FF}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs} \mathit{v2}. \text{getReport} (\text{prop} \mathit{v2}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs} \mathit{v1}. \text{getReport} (\text{notf} \mathit{v1}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs} \mathit{v4}. \text{getReport} (\mathit{v4} \text{ andf} \mathit{v5}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs} \mathit{v7} \mathit{v6}. \text{getReport} (\mathit{v6} \mathit{orf} \mathit{v7}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]

\[ (\forall \mathit{xs} \mathit{v8}. \text{getReport} (\mathit{v8} \text{ impf} \mathit{v9}::\mathit{xs}) = \text{getReport} \mathit{xs}) \land \]
\( \forall x_0 \ v_0, \ \text{getReport} (v_0 \ \text{eqf} \ v_0 :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0 \ v_1, \ \text{getReport} (v_1 \ \text{says} \ TT :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0 \ v_2, \ \text{getReport} (v_2 \ \text{says} \ FF :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0 \ v_34. \) 
\text{getReport} (\text{Name} v_34 \ \text{says} \ \text{prop} \ \text{NONE} :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0 \ v_46. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (ESC} v_46))) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL receiveMission))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL warno))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL tentativePlan))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL recon))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL completePlan))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL opoid))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL supervise))}) :: x_0) = \text{getReport} x_0 \) \land \\
\( \forall x_0. \) 
\text{getReport} (\text{Name PlatoonLeader} \ \text{says} \ \text{prop} \ (\text{SOME (SLc (PL report2))}) :: x_0) = \text{getReport} x_0 \) \land 

99
(∀ xs.
  getReport
  (Name PlatoonLeader says
   prop (SOME (SLc (PL complete))))::xs) =
  getReport xs) ∧
(∀ xs.
  getReport
  (Name PlatoonLeader says
   prop (SOME (SLc (PL plIncomplete))))::xs) =
  getReport xs) ∧
(∀ xs v151.
  getReport
  (Name PlatoonLeader says prop (SOME (SLc (PSG v151))))::xs) =
  getReport xs) ∧
(∀ xs v144.
  getReport
  (Name PlatoonSergeant says prop (SOME v144)::xs) =
  getReport xs) ∧
(∀ xs v68 v136 v137 v138.
  getReport
  (v137 quoting v138 says prop v68::xs) =
  getReport xs) ∧
(∀ xs v69 v12.
  getReport (v12 says notf v69::xs) = getReport xs) ∧
(∀ xs v71 v70 v12.
  getReport (v12 says (v70 andf v71)::xs) = getReport xs) ∧
(∀ xs v73 v72 v12.
  getReport (v12 says (v72 orf v73)::xs) = getReport xs) ∧
(∀ xs v75 v74 v12.
  getReport (v12 says (v74 impf v75)::xs) = getReport xs) ∧
(∀ xs v77 v76 v12.
  getReport (v12 says (v76 eqf v77)::xs) = getReport xs) ∧
(∀ xs v80 v12.
  getReport (v12 says v80 speaks_for v81::xs) =
  getReport xs) ∧
(∀ xs v82 v12.
  getReport (v12 says v82 controls v83::xs) =
  getReport xs) ∧
(∀ xs v86 v85 v84 v12.)
getReport (v_{12} \text{ says } \text{reps } v_{34}, v_{83}, v_{86}::xs) =
getReport xs \land
(\forall z s v_{28}, v_{87}, v_{12}.
getReport (v_{12} \text{ says } v_{87} \text{ domi } v_{88}::xs) = \text{getReport } xs) \land
(\forall z s v_{29}, v_{90}, v_{12}.
getReport (v_{12} \text{ says } v_{90} \text{ eqi } v_{90}::xs) = \text{getReport } xs) \land
(\forall z s v_{92}, v_{91}, v_{12}.
getReport (v_{12} \text{ says } v_{91} \text{ doms } v_{92}::xs) = \text{getReport } xs) \land
(\forall z s v_{94}, v_{93}, v_{12}.
getReport (v_{12} \text{ says } v_{93} \text{ eqs } v_{94}::xs) = \text{getReport } xs) \land
(\forall z s v_{96}, v_{95}, v_{12}.
getReport (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs) = \text{getReport } xs) \land
(\forall z s v_{98}, v_{97}, v_{12}.
getReport (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs) = \text{getReport } xs) \land
(\forall z s v_{99}, v_{12}, v_{100}.
getReport (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs) = \text{getReport } xs) \land
(\forall z s v_{15}, v_{14}.
getReport (v_{14} \text{ speaks_for } v_{15}::xs) = \text{getReport } xs) \land
(\forall z s v_{17}, v_{16}.
getReport (v_{16} \text{ controls } v_{17}::xs) = \text{getReport } xs) \land
(\forall z s v_{20}, v_{19}, v_{18}.
getReport (\text{reps } v_{18}, v_{19}, v_{20}::xs) = \text{getReport } xs) \land
(\forall z s v_{22}, v_{21}.
getReport (v_{21} \text{ domi } v_{22}::xs) = \text{getReport } xs) \land
(\forall z s v_{24}, v_{23}.
getReport (v_{23} \text{ eqi } v_{24}::xs) = \text{getReport } xs) \land
(\forall z s v_{26}, v_{25}.
getReport (v_{25} \text{ doms } v_{26}::xs) = \text{getReport } xs) \land
(\forall z s v_{28}, v_{27}.
getReport (v_{27} \text{ eqs } v_{28}::xs) = \text{getReport } xs) \land
(\forall z s v_{30}, v_{29}.
getReport (v_{29} \text{ eqn } v_{30}::xs) = \text{getReport } xs) \land
(\forall z s v_{32}, v_{31}.
getReport (v_{31} \text{ lte } v_{32}::xs) = \text{getReport } xs) \land
(\forall z s v_{34}, v_{33}.
getReport (v_{33} \text{ lt } v_{34}::xs) = \text{getReport } xs)

[getReport\_ind]
\begin{align*}
\vdash \forall P .
P & \land
(\forall z s .
\begin{align*}
(\text{Name PlatoonLeader says}
\quad \text{prop (SOME (SLc (PL report1)))::xs})) \land
\quad (\forall z s . \, P \, z s \to P (\text{TT::xs})) \land
\quad (\forall z s . \, P \, z s \to P (\text{FF::xs})) \land
\quad (\forall v_{2} \, x s . \, P \, x s \to P (\text{prop v}_{2}::xs)) \land
\quad (\forall v_{3} \, x s . \, P \, x s \to P (\text{notif v}_{3}::xs)) \land
\quad (\forall v_{4} \, v_{5} \, x s . \, P \, x s \to P (v_{4} \text{ andf } v_{5}::xs)) \land
\quad (\forall v_{6} \, v_{7} \, x s . \, P \, x s \to P (v_{6} \text{ orf } v_{7}::xs)) \land
\quad (\forall v_{8} \, v_{9} \, x s . \, P \, x s \to P (v_{8} \text{ impf } v_{9}::xs)) \land
\quad (\forall v_{10} \, v_{11} \, x s . \, P \, x s \to P (v_{10} \text{ eqf } v_{11}::xs)) \land
\quad (\forall v_{12} \, x s . \, P \, x s \to P (v_{12} \text{ says } \text{TT::xs})) \land
\quad (\forall v_{14} \, x s . \, P \, x s \to P (v_{14} \text{ says } \text{FF::xs})) \land
\quad (\forall v_{134} \, x s . \, P \, x s \to P (\text{Name v}_{134} \text{ says prop NONE::xs})) \land
\quad (\forall v_{146} \, x s .
P \, x s \to
\end{align*}
\end{align*}
\right)
}
\[
P\quad (\text{Name PlatoonLeader says prop (SOME (ESCc } v \text{)):xs}) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL receiveMission))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL warno))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL tentativePlan))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL recon))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL opoid))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL supervise))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL report2))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL complete))):xs})) \land \\
(\forall xs. \\
P\quad (\text{Name PlatoonLeader says} \\
\quad \text{prop (SOME (SLc (PL complete))):xs})) \land 
\]

102
\( P \; xs \Rightarrow \\
\quad (Name\; PlatoonLeader\; says \\
\quad \quad prop\; (SOME\; (SLc\; (PL\; plIncomplete))))::xs) \wedge \\
\forall\; xs. \quad P\; xs \Rightarrow \\
\quad (Name\; PlatoonLeader\; says \\
\quad \quad prop\; (SOME\; (SLc\; (PL\; invalidPlCommand))))::xs) \wedge \\
\forall\; v1\; xs. \quad P\; xs \Rightarrow \\
\quad (Name\; PlatoonLeader\; says \\
\quad \quad prop\; (SOME\; (SLc\; (PSG\; v1)))::xs) \wedge \\
\forall\; v14\; xs. \quad P\; xs \Rightarrow \\
\quad \quad P\; (Name\; PlatoonSergeant\; says \; prop\; (SOME\; v14)::xs) \wedge \\
\forall\; v135\; v136\; v8\; xs. \quad P\; xs \Rightarrow \; P\; (v135\; meet\; v136\; says\; prop\; v8::xs) \wedge \\
\forall\; v137\; v138\; v8\; xs. \quad P\; xs \Rightarrow \; P\; (v137\; quoting\; v138\; says\; prop\; v8::xs) \wedge \\
\forall\; v12\; v89\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; notf\; v89::xs) \wedge \\
\forall\; v12\; v70\; v71\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; (v70\; andf\; v71)::xs) \wedge \\
\forall\; v12\; v72\; v73\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; (v72\; orf\; v73)::xs) \wedge \\
\forall\; v12\; v74\; v75\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; (v74\; impf\; v75)::xs) \wedge \\
\forall\; v12\; v76\; v77\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; (v76\; eqf\; v77)::xs) \wedge \\
\forall\; v12\; v78\; v79\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v78\; says\; v79::xs) \wedge \\
\forall\; v12\; v80\; v81\; xs. \\
\quad P\; xs \Rightarrow \; P\; (v12\; says\; v80\; speaks_for\; v81::xs) \wedge \\
\forall\; v12\; v82\; v83\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v82\; controls\; v83::xs) \wedge \\
\forall\; v12\; v84\; v85\; v86\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; reps\; v84\; v85\; v86::xs) \wedge \\
\forall\; v12\; v87\; v88\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v87\; domi\; v88::xs) \wedge \\
\forall\; v12\; v89\; v90\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v89\; eqi\; v90::xs) \wedge \\
\forall\; v12\; v91\; v92\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v91\; doms\; v92::xs) \wedge \\
\forall\; v12\; v93\; v94\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v93\; eqs\; v94::xs) \wedge \\
\forall\; v12\; v95\; v96\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v95\; eqn\; v96::xs) \wedge \\
\forall\; v12\; v97\; v98\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v97\; lte\; v98::xs) \wedge \\
\forall\; v12\; v99\; v100\; xs. \quad P\; xs \Rightarrow \; P\; (v12\; says\; v99\; it\; v100::xs) \wedge \\
\forall\; v14\; v15\; xs. \quad P\; xs \Rightarrow \; P\; (v14\; speaks_for\; v15::xs) \wedge \\
\forall\; v16\; v17\; xs. \quad P\; xs \Rightarrow \; P\; (v16\; controls\; v17::xs) \wedge \\
\forall\; v18\; v19\; v20\; xs. \quad P\; xs \Rightarrow \; P\; (reps\; v18\; v19\; v20::xs) \wedge \\
\forall\; v21\; v22\; xs. \quad P\; xs \Rightarrow \; P\; (v21\; domi\; v22::xs) \wedge \\
\forall\; v23\; v24\; xs. \quad P\; xs \Rightarrow \; P\; (v23\; eqi\; v24::xs) \wedge \\
\forall\; v25\; v26\; xs. \quad P\; xs \Rightarrow \; P\; (v25\; doms\; v26::xs) \wedge \\
\forall\; v27\; v28\; xs. \quad P\; xs \Rightarrow \; P\; (v27\; eqs\; v28::xs) \wedge \\
\forall\; v29\; v30\; xs. \quad P\; xs \Rightarrow \; P\; (v29\; eqn\; v30::xs) \wedge \\
\forall\; v31\; v32\; xs. \quad P\; xs \Rightarrow \; P\; (v31\; lte\; v32::xs) \wedge
\[(\forall v_{33} \quad v_{34} \quad xs. \quad P \quad xs \quad \Rightarrow \quad P \quad (v_{33} \quad \text{lt} \quad v_{34}::xs)) \quad \Rightarrow \quad P \quad v \quad \]

\[\vdash \quad (\text{getTenativePlan} \quad [] \quad = \quad [\text{NONE}]) \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \]

\[\quad \text{Name} \quad \text{PlatoonLeader} \quad \text{says} \quad \]

\[\quad \text{prop} \quad \text{(SOME \quad (SLc \quad (PL \quad \text{tentativePlan})))::xs} \quad = \quad \]

\[\quad \text{(SOME \quad (SLc \quad (PL \quad \text{tentativePlan})))} \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{TT::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{FF::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{prop} \quad v_{2}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{notf} \quad v_{3}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad v_{4}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{prop} \quad v_{5}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{impf} \quad v_{9}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{orv} \quad v_{7}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{prop} \quad v_{8}::xs \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{prop} \quad \text{(SOME \quad (ESCc} \quad v_{134})::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{Name} \quad v_{134} \quad \text{says} \quad \text{prop} \quad \text{NONE::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{Name} \quad \text{PlatoonLeader} \quad \text{says} \quad \]

\[\quad \text{prop} \quad \text{(SOME \quad (SLc \quad (PL \quad \text{receiveMission})))::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{Name} \quad \text{PlatoonLeader} \quad \text{says} \quad \]

\[\quad \text{prop} \quad \text{(SOME \quad (SLc \quad (PL \quad \text{warno})))::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]

\[\forall xs. \quad \text{getTenativePlan} \quad \text{Name} \quad \text{PlatoonLeader} \quad \text{says} \quad \]

\[\quad \text{prop} \quad \text{(SOME \quad (SLc \quad (PL \quad \text{recon})))::xs} \quad = \quad \text{getTenativePlan} \quad xs \quad \land \quad \]
\[(\forall \mathbf{x}s, \text{getTenativePlan}) \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL report1))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL completePlan))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL opoid))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL supervise))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL report2))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL complete))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PL plIncomplete))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s \mathbf{v}151, \text{getTenativePlan} \quad \text{(Name PlatoonLeader says prop (SOME (S\mathcal{L}c (PSG \mathbf{v}151))))::xs}) = \\
\text{getTenativePlan} \quad \mathbf{x}s \land \\
(\forall \mathbf{x}s \mathbf{v}144, \text{getTenativePlan} \quad \text{(Name PlatoonSergeant says prop (SOME \mathbf{v}144)::xs}) = \]

105
\begin{align*}
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{136} \; v_{137} \; v_{135}.
& \text{getTenativePlan } (v_{137} \; \text{quoting } v_{138} \; \text{says } \text{prop } v_{08} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{136} \; v_{137}.
& \text{getTenativePlan } (v_{136} \; \text{meet } v_{137} \; \text{says } \text{prop } v_{08} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{09} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } \text{notf } v_{09} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{71} \; v_{70} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } (v_{70} \; \text{andf } v_{71}) :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{73} \; v_{72} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } (v_{72} \; \text{orf } v_{73}) :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{75} \; v_{74} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } (v_{74} \; \text{impf } v_{75}) :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{77} \; v_{76} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } (v_{76} \; \text{eqf } v_{77}) :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{79} \; v_{78} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{78} :: v_{79} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{84} \; v_{83} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{83} \; \text{speaks_for } v_{84} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{86} \; v_{85} \; v_{84} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } \text{reps } v_{84} \; v_{85} \; v_{86} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{88} \; v_{87} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{87} \; \text{domi } v_{88} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{90} \; v_{89} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{89} \; \text{eqi } v_{90} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{92} \; v_{91} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{91} \; \text{doms } v_{92} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{94} \; v_{93} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{93} \; \text{eqs } v_{94} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& (\forall xs \; v_{96} \; v_{95} \; v_{12}.
& \text{getTenativePlan } (v_{12} \; \text{says } v_{95} \; \text{eqn } v_{96} :: xs) = \\
& \text{getTenativePlan } xs) \land \\
& \end{align*}
\( \forall x s \, v_{98} \, v_{97} \, v_{12}. \)
\( \text{getTenativePlan} \ (v_{12} \ \text{says} \ v_{97} \ \text{lte} \ v_{98}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{99} \, v_{12} \, v_{100}. \)
\( \text{getTenativePlan} \ (v_{12} \ \text{says} \ v_{99} \ \text{lt} \ v_{100}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{15} \, v_{14}. \)
\( \text{getTenativePlan} \ (v_{14} \ \text{speaks_for} \ v_{15}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{17} \, v_{16}. \)
\( \text{getTenativePlan} \ (v_{16} \ \text{controls} \ v_{17}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{20} \, v_{19} \, v_{18}. \)
\( \text{getTenativePlan} \ (\text{reps} \ v_{18} \ v_{19} \ v_{20}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{22} \, v_{21}. \)
\( \text{getTenativePlan} \ (v_{21} \ \text{domi} \ v_{22}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{24} \, v_{23}. \)
\( \text{getTenativePlan} \ (v_{24} \ \text{eqi} \ v_{24}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{26} \, v_{25}. \)
\( \text{getTenativePlan} \ (v_{25} \ \text{doms} \ v_{26}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{28} \, v_{27}. \)
\( \text{getTenativePlan} \ (v_{27} \ \text{eqs} \ v_{28}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{30} \, v_{29}. \)
\( \text{getTenativePlan} \ (v_{29} \ \text{eqn} \ v_{30}::xs) = \text{getTenativePlan} \ xs \) \(
\land \ (\forall x s \, v_{32} \, v_{31}. \)
\( \text{getTenativePlan} \ (v_{31} \ \text{lte} \ v_{32}::xs) = \text{getTenativePlan} \ xs \) \(\forall x s \, v_{34} \, v_{33}. \)
\( \text{getTenativePlan} \ (v_{33} \ \text{lt} \ v_{34}::xs) = \text{getTenativePlan} \ xs \)

[getTenativePlan_ind]

\( \vdash \ \forall P. \)
\( \vdash \ (\forall x s. \ P) \land \)
\( (\text{Name PlatoonLeader says}) \)
\( \text{prop} \ (\text{SOME} \ (\text{SLc} \ (\text{PL tentativePlan}))::xs)) \land \)
\( (\forall x s. \ P \ xs \ \Rightarrow \ P \ (\text{TT}::xs)) \land \ (\forall x s. \ P \ xs \ \Rightarrow \ P \ (\text{FF}::xs)) \land \)
\( (\forall v_{2} \ x s. \ P \ xs \ \Rightarrow \ P \ (\text{prop} \ v_{2}::xs)) \land \)
\( (\forall v_{3} \ x s. \ P \ xs \ \Rightarrow \ P \ (\text{notf} \ v_{3}::xs)) \land \)
\( (\forall v_{4} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{4} \ \text{andf} \ v_{5}::xs)) \land \)
\( (\forall v_{6} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{6} \ \text{orf} \ v_{7}::xs)) \land \)
\( (\forall v_{8} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{8} \ \text{impf} \ v_{9}::xs)) \land \)
\( (\forall v_{10} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{10} \ \text{eqf} \ v_{11}::xs)) \land \)
\( (\forall v_{12} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{12} \ \text{sayf} \ v_{12}::xs)) \land \)
\( (\forall v_{13} \ x s. \ P \ xs \ \Rightarrow \ P \ (v_{13} \ \text{sayf} \ v_{14}::xs)) \land \)
\( (\forall v_{14} \ x s. \ P \ xs \ \Rightarrow \ P \ (\text{Name} \ v_{14} \ \text{sayf} \ v_{15}::xs)) \land \)
\( (\forall v_{14} \ x s. \ P \ xs \ \Rightarrow \ P \ (\text{Name} \ v_{14} \ \text{sayf} \ v_{16}::xs)) \land \)
\( P \ xs \Rightarrow \)
\(\frac{P}{\text{Name PlatoonLeader says prop \(\text{SOME (ESCc \(v146\))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL receiveMission))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL warno))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL recon))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL report1))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL completePlan))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL opoid))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL supervise))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL report2))::xs})\} \land}
\end{equation}

\(\forall xs, P \Rightarrow P \text{Name PlatoonLeader says prop \(\text{SOME (SLc (PL complete))::xs})\} \land}
\end{equation}
Theorems

\[ P \quad xs \Rightarrow \]
\[ P \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL plIncomplete))::xs))} \land \]
\[ \forall \, xs. \]
\[ P \quad xs \Rightarrow \]
\[ P \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PL invalidPlCommand))::xs))} \land \]
\[ \forall \, v_{151} \, xs. \]
\[ P \quad xs \Rightarrow \]
\[ P \]
\[ \text{Name PlatoonLeader says} \]
\[ \text{prop (SOME (SLc (PSG v_{151}))::xs))} \land \]
\[ \forall \, v_{144} \, xs. \]
\[ P \quad xs \Rightarrow \]
\[ P \quad (\text{Name PlatoonSergeant says prop (SOME \: v_{144})::xs))} \land \]
\[ \forall \, v_{135} \quad v_{136} \quad v_{98} \: xs. \]
\[ P \quad xs \Rightarrow \quad P \quad (v_{135} \quad \text{meet} \quad v_{136} \quad \text{says prop} \quad v_{98}::xs))} \land \]
\[ \forall \, v_{137} \quad v_{138} \quad v_{98} \: xs. \]
\[ P \quad xs \Rightarrow \quad P \quad (v_{137} \quad \text{quoting} \quad v_{138} \quad \text{says prop} \quad v_{98}::xs))} \land \]
\[ \forall \, v_{12} \quad v_{9} \quad v_{99} \: xs. \quad P \quad xs \Rightarrow \quad P \quad (v_{12} \quad \text{says} \quad v_{99}::xs))} \land \]
\[ \forall \, v_{12} \quad v_{70} \quad v_{71} \quad v_{72} \quad v_{73} \quad v_{74} \quad v_{75} \quad v_{76} \quad v_{77} \quad v_{78} \quad v_{79} \quad v_{90} \quad v_{91} \quad v_{92} \quad v_{93} \quad v_{94} \quad v_{95} \quad v_{96} \quad v_{97} \quad v_{98} \quad v_{99} \quad \text{xs.} \]
\[ P \quad xs \Rightarrow \quad P \quad (v_{12} \quad \text{says} \quad v_{99} \quad \text{speak_for} \quad v_{98}::xs))} \land \]
\[ \forall \, v_{12} \quad v_{92} \quad v_{93} \quad v_{94} \quad v_{95} \quad v_{96} \quad v_{97} \quad v_{98} \quad v_{99} \quad v_{100} \quad \text{xs.} \quad P \quad xs \Rightarrow \quad P \quad (v_{12} \quad \text{says} \quad v_{99} \quad \text{lt} \quad v_{100}::xs))} \land \]
\[ \forall \, v_{14} \quad v_{15} \quad v_{16} \quad v_{17} \quad v_{18} \quad v_{19} \quad v_{20} \quad v_{21} \quad v_{22} \quad v_{23} \quad v_{24} \quad v_{25} \quad v_{26} \quad v_{27} \quad v_{28} \quad v_{29} \quad v_{30} \quad v_{31} \quad v_{32} \quad v_{33} \quad v_{34} \quad v_{35} \quad v_{36} \quad v_{37} \quad v_{38} \quad v_{39} \quad v_{40} \quad v_{41} \quad v_{42} \quad v_{43} \quad v_{44} \quad v_{45} \quad v_{46} \quad v_{47} \quad v_{48} \quad v_{49} \quad v_{50} \quad v_{51} \quad v_{52} \quad v_{53} \quad v_{54} \quad v_{55} \quad v_{56} \quad v_{57} \quad v_{58} \quad v_{59} \quad v_{60} \quad v_{61} \quad v_{62} \quad v_{63} \quad v_{64} \quad v_{65} \quad v_{66} \quad v_{67} \quad v_{68} \quad v_{69} \quad v_{70} \quad v_{71} \quad v_{72} \quad v_{73} \quad v_{74} \quad v_{75} \quad v_{76} \quad v_{77} \quad v_{78} \quad v_{79} \quad v_{80} \quad v_{81} \quad v_{82} \quad v_{83} \quad v_{84} \quad v_{85} \quad v_{86} \quad v_{87} \quad v_{88} \quad v_{89} \quad v_{90} \quad v_{91} \quad v_{92} \quad v_{93} \quad v_{94} \quad v_{95} \quad v_{96} \quad v_{97} \quad v_{98} \quad v_{99} \quad v_{100} \quad \text{xs.} \quad P \quad xs \Rightarrow \quad P \quad (v_{14} \quad \text{speak_for} \quad v_{15}::xs))} \land \]
\[ \forall \, v_{16} \quad v_{17} \quad v_{18} \quad v_{19} \quad v_{20} \quad v_{21} \quad v_{22} \quad v_{23} \quad v_{24} \quad v_{25} \quad v_{26} \quad v_{27} \quad v_{28} \quad v_{29} \quad v_{30} \quad v_{31} \quad v_{32} \quad v_{33} \quad v_{34} \quad v_{35} \quad v_{36} \quad v_{37} \quad v_{38} \quad v_{39} \quad v_{40} \quad v_{41} \quad v_{42} \quad v_{43} \quad v_{44} \quad v_{45} \quad v_{46} \quad v_{47} \quad v_{48} \quad v_{49} \quad v_{50} \quad v_{51} \quad v_{52} \quad v_{53} \quad v_{54} \quad v_{55} \quad v_{56} \quad v_{57} \quad v_{58} \quad v_{59} \quad v_{60} \quad v_{61} \quad v_{62} \quad v_{63} \quad v_{64} \quad v_{65} \quad v_{66} \quad v_{67} \quad v_{68} \quad v_{69} \quad v_{70} \quad v_{71} \quad v_{72} \quad v_{73} \quad v_{74} \quad v_{75} \quad v_{76} \quad v_{77} \quad v_{78} \quad v_{79} \quad v_{80} \quad v_{81} \quad v_{82} \quad v_{83} \quad v_{84} \quad v_{85} \quad v_{86} \quad v_{87} \quad v_{88} \quad v_{89} \quad v_{90} \quad v_{91} \quad v_{92} \quad v_{93} \quad v_{94} \quad v_{95} \quad v_{96} \quad v_{97} \quad v_{98} \quad v_{99} \quad v_{100} \quad \text{xs.} \quad P \quad xs \Rightarrow \quad P \quad (v_{16} \quad \text{controls} \quad v_{17}::xs))} \land \]

109
(∀ v₃₃ v₃₄ xs. P xs ⇒ P (v₃₃ α v₃₄::xs)) ⇒
∀ v. P v
Index

ConductORPDef Theory, 46
Definitions, 46
secAuthorization_def, 46
secContext_def, 46
secHelper_def, 47
Theorems, 47
getOmniCommand_def, 47
getOmniCommand_ind, 49
getPlCom_def, 50
getPlCom_ind, 51
getPsgCom_def, 51
getPsgCom_ind, 51

ConductORPType Theory, 44
Datatypes, 44
Theorems, 45
omniCommand_distinct_clauses, 45
plCommand_distinct_clauses, 45
psgCommand_distinct_clauses, 45
slCommand_distinct_clauses, 45
slCommand_one_one, 45
slOutput_distinct_clauses, 45
slRole_distinct_clauses, 46
slState_distinct_clauses, 46

ConductPBType Theory, 57
Datatypes, 57
Theorems, 57
plCommandPB_distinct_clauses, 57
psgCommandPB_distinct_clauses, 57
slCommand_distinct_clauses, 57
slCommand_one_one, 57
slOutput_distinct_clauses, 57
slRole_distinct_clauses, 58
slState_distinct_clauses, 58

MoveToORPType Theory, 62
Datatypes, 62
Theorems, 63
slCommand_distinct_clauses, 63
slOutput_distinct_clauses, 63
slState_distinct_clauses, 63

MoveToPBType Theory, 68
Datatypes, 68
Theorems, 68
slCommand_distinct_clauses, 68
slOutput_distinct_clauses, 68
slState_distinct_clauses, 69

OMNIType Theory, 3
Datatypes, 3
Theorems, 3
command_distinct_clauses, 3
command_one_one, 3
escCommand_distinct_clauses, 3
escOutput_distinct_clauses, 3
escState_distinct_clauses, 3
output_distinct_clauses, 4
output_one_one, 4
principal_one_one, 4
state_distinct_clauses, 4
state_one_one, 4

PBIntegratedDef Theory, 23
Definitions, 23
secAuthorization_def, 23
secContext_def, 23
secHelper_def, 24
Theorems, 24
getOmniCommand_def, 24
getOmniCommand_ind, 27
getPlCom_def, 28
getPlCom_ind, 28

PBTypeIntegrated Theory, 21
Datatypes, 21
Theorems, 22
omniCommand_distinct_clauses, 22
plCommand_distinct_clauses, 22
slCommand_distinct_clauses, 22
slCommand_one_one, 22
slOutput_distinct_clauses, 23
slState_distinct_clauses, 23
stateRole_distinct_clauses, 23

PlanPBDef Theory, 82
Definitions, 82
   PL_notWARNO_Auth_def, 82
   PL_WARNO_Auth_def, 82
   secContext_def, 82
   secContextNull_def, 83
Theorems, 83
   getInitMove_def, 83
   getInitMove_ind, 85
   getPlCom_def, 87
   getPlCom_ind, 89
   getPsgCom_def, 90
   getPsgCom_ind, 92
   getRecon_def, 93
   getRecon_ind, 96
   getReport_def, 98
   getReport_ind, 101
   getTentativePlan_def, 104
   getTentativePlan_ind, 107

PlanPBType Theory, 79
Datatypes, 79
Theorems, 79
   plCommand_distinct_clauses, 79
   psgCommand_distinct_clauses, 80
   slCommand_distinct_clauses, 80
   slCommand_one_one, 80
   slOutput_distinct_clauses, 80
   slRole_distinct_clauses, 81
   slState_distinct_clauses, 81

satList Theory, 21
Definitions, 21
   satList_def, 21
Theorems, 21
   satList_conj, 21
   satList_CONS, 21
   satList_nil, 21

ssm Theory, 11
Datatypes, 11
Definitions, 12
   authenticationTest_def, 12
   commandList_def, 12
   inputList_def, 12
   propCommandList_def, 12
   TR_def, 12
Theorems, 13
   CFGInterpret_def, 13
   CFGInterpret_ind, 13
   configuration_one_one, 13
   extractCommand_def, 13
   extractCommand_ind, 13
   extractInput_def, 14
   extractInput_ind, 14
   extractPropCommand_def, 15
   extractPropCommand_ind, 15
   TR_cases, 16
   TR_discard_cmd_rule, 17
   TR_EQ_rules_thm, 17
   TR_exec_cmd_rule, 17
   TR_ind, 18
   TR_rules, 18
   TR_strongind, 19
   TR_trap_cmd_rule, 20
   TRrule0, 20
   TRrule1, 20
   trType_distinct_clauses, 20
   trType_one_one, 21

ssm11 Theory, 4
Datatypes, 4
Definitions, 4
   TR_def, 4
Theorems, 5
   CFGInterpret_def, 5
   CFGInterpret_ind, 6
   configuration_one_one, 6
   order_distinct_clauses, 6
   order_one_one, 6
   TR_cases, 6
   TR_discard_cmd_rule, 7
   TR_EQ_rules_thm, 7
   TR_exec_cmd_rule, 8
   TR_ind, 8
   TR_rules, 9
   TR_strongind, 9
   TR_trap_cmd_rule, 10
   TRrule0, 10
   TRrule1, 11
   trType_distinct_clauses, 11
ssmConductORP Theory, 35
Definitions, 35
Theorems, 35
conductORPNS_def, 35
conductORPNS_ind, 35
conductORPOut_def, 36
conductORPOut_ind, 36
inputOK_cmd_reject_lemma, 36
inputOK_def, 36
inputOK_ind, 37
PlatoonLeader_ACTIONS_IN_exec_justified_lemma, 38
PlatoonLeader_ACTIONS_IN_exec_justified_thm, 39
PlatoonLeader_ACTIONS_IN_exec_lemma, 39
PlatoonLeader_ACTIONS_INTrap_justified_lemma, 40
PlatoonLeader_ACTIONS_INTrap_justified_thm, 41
PlatoonLeader_ACTIONS_INTrap_lemma, 41
PlatoonLeader_CONDUCT_ORP_exec_secure_justified_lemma, 42
PlatoonLeader_CONDUCT_ORP_exec_secure_thm, 42
PlatoonLeader_CONDUCT_ORP_exec_lemma, 42
PlatoonSergeant_SECURE_exec_justified_lemma, 44
PlatoonSergeant_SECURE_exec_justified_thm, 43
PlatoonSergeant_SECURE_exec_lemma, 44

ssmMoveToORP Theory, 58
Definitions, 58
secContextMoveToORP_def, 58
ssmMoveToORPStateInterp_def, 58
Theorems, 58
authTestMoveToORP_cmd_reject_lemma, 58
authTestMoveToORP_def, 58
authTestMoveToORP_ind, 59
moveToORPNS_def, 60
moveToORPNS_ind, 60
moveToORPOut_def, 61
moveToORPOut_ind, 61
PlatoonLeader_exec_slCommand_justified_thm, 62
PlatoonLeader_exec_slCommand_lemma, 62

ssmConductPB Theory, 51
Definitions, 51
secContextConductPB_def, 51
ssmConductPBStateInterp_def, 52
Theorems, 52
authTestConductPB_cmd_reject_lemma, 52
authTestConductPB_def, 52
authTestConductPB_ind, 53
conductPBNS_def, 53
conductPBNS_ind, 54
conductPBOut_def, 54
conductPBOut_ind, 55
PlatoonLeader_exec_plCommandPB_justified_thm, 55
PlatoonLeader_plCommandPB_lemma, 56
PlatoonSergeant_exec_psgCommandPB_justified_thm, 56
PlatoonSergeant_psgCommandPB_lemma, 56

ssmMoveToPB Theory, 63
Definitions, 63
secContextMoveToPB_def, 63
ssmMoveToPBStateInterp_def, 64
Theorems, 64
authTestMoveToPB_cmd_reject_lemma, 64
authTestMoveToPB_def, 64
authTestMoveToPB_ind, 65
moveToPBNS_def, 65
moveToPBNS_ind, 66
moveToPBOut_def, 66
moveToPBOut_ind, 66
PlatoonLeader_exec_slCommand_justified_thm, 67
PlatoonLeader_slCommand_lemma, 68

ssmPBIntegrated Theory, 28
Theorems, 28
inputOK_cmd_reject_lemma, 28
inputOK_def, 28
inputOK_ind, 29
PBNS_def, 30
PBNS_ind, 30
PBOOut_def, 30
PBOOut_ind, 31
PlatoonLeader_Omni_notDiscard_slCommand_thm, 31
PlatoonLeader_PLAN_PB_exec_justified_lemma, 31
PlatoonLeader_PLAN_PB_exec_justified_thm, 32
PlatoonLeader_PLAN_PB_exec_lemma, 33
PlatoonLeader_PLAN_PB_trap_justified_lemma, 33
PlatoonLeader_PLAN_PB_trap_justified_thm, 34
PlatoonLeader_PLAN_PB_trap_lemma, 35

ssmPlanPB Theory, 69
Theorems, 69
inputOK_def, 69
inputOK_ind, 70
planPBNS_def, 70
planPBNS_ind, 71
planPBOOut_def, 71
planPBOOut_ind, 72
PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma, 72
PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm, 73
PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma, 73
PlatoonLeader_psgCommand_notDiscard_thm, 74
PlatoonLeader_trap_psgCommand_justified_lemma, 74
PlatoonLeader_trap_psgCommand_lemma, 74
PlatoonLeader_WARNO_exec_report1_justified_lemma, 75
PlatoonLeader_WARNO_exec_report1_justified_thm, 76
PlatoonLeader_WARNO_exec_report1_lemma, 77
PlatoonSergeant_trap_plCommand_justified_lemma, 77
PlatoonSergeant_trap_plCommand_justified_thm, 78
PlatoonSergeant_trap_plCommand_lemma, 78

114
Appendix C

Parametrizable Secure State Machine Theories: HOL Script Files

C.1 ssm

(* Secure State Machine Theory: authentication, authorization, and state *)
(* interaction. *)
(* Author: Shiu-Kai Chin *)
(* Date: 27 November 2015 *)

structure ssmScript = struct

(* Interactive mode *)
app load ["TypeBase", "ssminfRules", "listTheory", "optionTheory", "acl_infRules", "satListTheory", "ssmTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheory ssmTheory

app load ["TypeBase", "ssminfRules", "listTheory", "optionTheory", "acl_infRules", "satListTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheory ssmTheory

(* end interactive mode *)

open HolKernel boolLib Parse bossLib
open TypeBase listTheory optionTheory ssminfRules acl_infRules satListTheory

(* create a new theory *)

val _ = new_theory "ssm";

(* Define the type of transition: discard, execute, or trap. We discard from *)
(* the input stream those inputs that are not of the form P says command. We *)
(* execute commands that users and supervisors are authorized for. We trap *)
(* commands that users are not authorized to execute. *)

(* In keeping with virtual machine design principles as described by Popek *)
(* and Goldberg, we add a TRAP instruction to the commands by users. *)
(* In effect, we are LIFTING the commands available to users to include the *)
(* TRAP instruction used by the state machine to handle authorization errors. *)
val _ =
Datatype
  'trType =
    discard 'cmdlist | trap 'cmdlist | exec 'cmdlist
val trType_distinct_clauses = distinct_of }):'cmdlist trType`
val _ = save_thm("trType_distinct_clauses",trType_distinct_clauses)
val trType_one_one = one_one_of }):'cmdlist trType`
val _ = save_thm("trType_one_one",trType_one_one)

(* Define configuration to include the security context within which the *)
(* inputs are evaluated. The components are as follows: (1) the authentica*)
(* tion function, (2) the interpretation of the state, (3) the security context, *)
(* (4) the input stream, (5) the state, and (6) the output stream. *)
val _ =
Datatype
  'configuration =
CFG
  ((('command option , 'principal , 'd, 'e)Form -> bool)
  (('state -> (('command option , 'principal , 'd, 'e)Form list ->
     (('command option , 'principal , 'd, 'e)Form list))
  ((('command option , 'principal , 'd, 'e)Form list))
  (((('command option , 'principal , 'd, 'e)Form list) list)
       ('state)
       ('output list))

(* Prove one-to-one properties of configuration *)
val configuration_one_one =
  one_one_of }):('command option , 'd, 'e, 'output , 'principal , 'state)configuration`'
val _ = save_thm("configuration_one_one",configuration_one_one)

(* The interpretation of configuration is the conjunction of the formulas in *)
(* the context and the first element of a non-empty input stream. *)
val CFGInterpret_def =
Define
  CFGInterpret
    ((M:('command option , 'b, 'principal , 'd, 'e)Kripke),Oi:'d po,Os:'e po)
    (CFG
     (elementTest:('command option , 'principal , 'd, 'e)Form -> bool)
     (CFG
      (elementTest:('command option , 'principal , 'd, 'e)Form list ->
       ((('command option , 'principal , 'd, 'e)Form list))
      ((('command option , 'principal , 'd, 'e)Form list))
      (((('command option , 'principal , 'd, 'e)Form list) list)
       (state)
       (output list))
      (cmdlist | trap 'cmdlist | exec 'cmdlist)
      (stateInterp:state -> ((('command option , 'principal , 'd, 'e)Form list) ->
       ((('command option , 'principal , 'd, 'e)Form list))
       ((('command option , 'principal , 'd, 'e)Form list))
       (((('command option , 'principal , 'd, 'e)Form list) list):ins)
       (state:state)
       (outStream:'output list))
    =
    ((M,Oi,Os) satList (context x)) /
    ((M,Oi,Os) satList x) /
    ((M,Oi,Os) satList (stateInterp state x))`'

(* In the following definitions of authenticationTest , extractCommand, and *)
(* commandList, we implicitly assume that the only authenticated inputs are *)
(* of the form P says phi, i.e., we know who is making statement phi. *)
("Simple proofs of security properties")
val authenticationTest_def =
Define
  'authenticationTest
    (elementTest:('command option , 'principal , 'd, 'e)Form -> bool)
    (x:('command option , 'principal , 'd, 'e)Form list) =
    FOLDR (\ p q.p \ q) T (MAP elementTest x)";
val extractCommand_def = Define
'extractCommand (P says (prop (SOME cmd))):('command option , 'principal , 'd, 'e)Form = cmd';

val commandList_def = Define
'commandList (x:('command option , 'principal , 'd, 'e)Form list) = MAP extractCommand x';

val extractPropCommand_def = Define
'(extractPropCommand (P says (prop (SOME cmd))):('command option , 'principal , 'd, 'e)Form) =
((prop (SOME cmd)):'command option , 'principal , 'd, 'e)Form)';

val propCommandList_def = Define
'propCommandList (x:('command option , 'principal , 'd, 'e)Form list) = MAP extractPropCommand x';

val extractInput_def = Define
'extractInput (P says (prop x):(command option , 'principal , 'd, 'e)Form) = x';

val inputList_def = Define
'inputList (xs:(command option , 'principal , 'd, 'e)Form list) = MAP extractInput xs';

('Define transition relation among configurations. This definition is
parameterized in terms of next-state transition function and output
function.

val (TR_rules, TR_ind, TR_cases) = Hol_reln
'!(elementTest:('command option , 'principal , 'd, 'e)Form -> bool)
(NS: 'state -> (command option list) trType -> 'state) M Oi Os Out (s:'state)
(context:((('command option , 'principal , 'd, 'e)Form list) ->
('command option , 'principal , 'd, 'e)Form list))
(stateInterp:'state -> ((command option , 'principal , 'd, 'e)Form list) ->
('command option , 'principal , 'd, 'e)Form list))
(ins:(('command option , 'principal , 'd, 'e)Form list list)
(outs:'output list).
(authenticationTest elementTest x) /
(CFGInterpret (M,Oi,Os)
(CFG elementTest stateInterp context (x::ins) s outs)) =>
(TR
((M:('command option , 'b, 'principal , 'd, 'e)Kripke),Oi : 'd po, Os: 'e po)
(exec (inputList x))
(CFG elementTest stateInterp context (x::ins) s outs)
(CFG elementTest stateInterp context ins
(NS s (exec (inputList x))))
(authenticationTest elementTest x) /
(CFGInterpret (M,Oi,Os)
(CFG elementTest stateInterp context (x::ins) s outs)) =>
(TR
((M:('command option , 'b, 'principal , 'd, 'e)Kripke),Oi : 'd po, Os: 'e po)
(trap (inputList x))
(CFG elementTest stateInterp context (x::ins) s outs)
(CFG elementTest stateInterp context ins

325
\[\{(\text{elementTest}::('command option,'principal,'d,'e)Form}\rightarrow\text{bool})\]
\{(\text{state}::('command option list) trType::'state) M O i O s\}
\{(\text{context}::('command option,'principal,'d,'e)Form list)\}
\{(\text{stateInterp}::('command option,'principal,'d,'e)Form list)\}
\{(\text{ins}::('command option,'principal,'d,'e)Form list list)\}
\{(\text{outs}::output list)\}
\]
\(\text{TR}((\text{M}::('command option,'b,'principal,'d,'e)Kripke),\text{O}::'dp,\text{Os}::'e po)\)
\(\text{CFG} \text{elementTest} \text{stateInterp context} (x::\text{ins}) \text{ s outs})\)
\(\text{CFG} \text{elementTest} \text{stateInterp context} \text{ ins}
\{(\text{NS s (discard (inputList x)))\}
\{(\text{Out s (discard (inputList x))):: outs}))\}

\text{val} \ [\text{rule0, rule1, rule2}] = \text{CONJUNCTS TR\_rules}

\(*\begin{align*}
\text{val} \ TR\_lemma0 &= \text{TAC\_PROOF}(([]), \text{flip\_TR\_rules rule0}), \\\n&\text{DISCH\_TAC\ THEN} \\\n&\text{IMP\_RES\ TAC TR\_cases\ THEN} \\
&\text{PAT\_ASSUM} \\
&\text{"exec cmd} = \text{y}" \\\n&\text{(fn th} \Rightarrow \text{ASSUME\_TAC(REWRITE\_RULE}[\text{trType\_one\_one, trType\_distinct\_clauses}\mid\text{th})] \text{THEN} \\
&\text{PROVE\_TAC}[\text{configuration\_one\_one, list\_11, trType\_distinct\_clauses}\]} \\
\end{align*}\)

\text{val} \ TR\_lemma1 = \text{TAC\_PROOF}(([]), \text{flip\_TR\_rules rule1}), \\\n\text{DISCH\_TAC\ THEN} \\\n\text{IMP\_RES\ TAC TR\_cases\ THEN} \\\n\text{PAT\_ASSUM} \\
\text{"trap cmd} = \text{y}" \\
\text{(fn th} \Rightarrow \text{ASSUME\_TAC(REWRITE\_RULE}[\text{trType\_one\_one, trType\_distinct\_clauses}\mid\text{th})] \text{THEN} \\
\text{PROVE\_TAC}[\text{configuration\_one\_one, list\_11, trType\_distinct\_clauses}\]} \\

\text{val} \ TR\_lemma2 = \text{TAC\_PROOF}(([]), \text{flip\_TR\_rules rule2}), \\\n\text{DISCH\_TAC\ THEN} \\\n\text{IMP\_RES\ TAC TR\_cases\ THEN} \\\n\text{PAT\_ASSUM} \\
\text{"discard (inputList x) = y"} \\\n\text{(fn th} \Rightarrow \text{ASSUME\_TAC(REWRITE\_RULE}[\text{trType\_one\_one, trType\_distinct\_clauses}\mid\text{th})] \text{THEN} \\
\text{PROVE\_TAC}[\text{configuration\_one\_one, list\_11, trType\_distinct\_clauses}\]} \\

\text{val} \ TR\_rules\_converse = \text{TAC\_PROOF}(([]), \text{flip\_TR\_rules TR\_rules}), \\\n\text{REWRITE\_TAC[TR\_lemma0, TR\_lemma1, TR\_lemma2]} \\
\text{val} \ TR\_EQ\_rules\_thm = \text{TR\_EQ\_rules TR\_rules TR\_rules\_converse} \\
\text{val} \ = \text{save\_thm("TR\_EQ\_rules\_thm","TR\_EQ\_rules\_thm")} \\
\text{val} \ [\text{TRrule0, TRrule1, TR\_discard\_cmd\_rule}] = \text{CONJUNCTS TR\_EQ\_rules\_thm} \\
\text{val} \ = \text{save\_thm("TRrule0",TRrule0)} \\
\text{val} \ = \text{save\_thm("TRrule1",TRrule1)} \\
\text{val} \ = \text{save\_thm("TR\_discard\_cmd\_rule",TR\_discard\_cmd\_rule)} \\

\text{val} \ [\text{TRrule0, TRrule1, TR\_discard\_cmd\_rule}] = \text{CONJUNCTS TR\_EQ\_rules\_thm} \\
\text{val} \ = \text{save\_thm("TRrule0",TRrule0)} \\
\text{val} \ = \text{save\_thm("TRrule1",TRrule1)} \\
\text{val} \ = \text{save\_thm("TR\_discard\_cmd\_rule",TR\_discard\_cmd\_rule)}
\[(M, O_i, O_s) \text{ sat } (\text{prop NONE}) \implies
\]
\[(\text{INS Out} \ M O_i O_s).
\]
TR
\[
(M : ('\text{command option}', 'b', '\text{principal}', 'd', 'e') \text{ Kripke}), (O_i : 'd \text{ po}),
(O_s : 'e \text{ po})) \implies \text{ trap (inputList x)}
\]
\[
(\text{CFG} \text{ elementTest} : ('\text{command option}', '\text{principal}', 'd', 'e') \text{ Form } \text{ bool})
\]
\[
\text{(stateInterp : 'state } \implies (\text{command option}', '\text{principal}', 'd', 'e') \text{ Form list } \implies
\]
\[
(\text{context } : (\text{command option}', '\text{principal}', 'd', 'e') \text{ Form list } \implies
\]
\[
(\text{x : : ins})
\]
\[
(s : 'state) \implies (\text{outs } : \text{output list})
\]
\[
(\text{CFG elementTest stateInterp context ins}
\]
\[
((\text{INS : 'state } \implies (\text{command option list trType } \implies 'state) \text{ s } \text{ (trap (inputList x))})
\]
\[
(\text{Out s } \text{ (trap (inputList x))}:: \text{outs}) \iff
\]
\[
(\text{authenticationTest elementTest x}) \land
\]
\[
(\text{CFGInterpret (M, O_i, O_s)}
\]
\[
(\text{CFG elementTest stateInterp context (x : : ins) s outs}) \land
\]
\[
(\text{(M, O_i, O_s) sat (prop NONE)})''),
\]
\[
\text{REWRITE_TAC [ TR rule1 ] THEN}
\]
\[
\text{REPEAT STRIP_TAC THEN}
\]
\[
\text{EQ_TAC THEN}
\]
\[
\text{REPEAT STRIP_TAC THEN}
\]
\[
\text{PROVE_TAC [ ]}
\]
\[
\text{val _ = save_thm ("TR_trap_cmd_rule",TR_trap_cmd_rule)}
\]
\[
\text{(* ===== start here =====}
\]
\[
\text{==== end here ===== = *)}
\]
\[
\text{val _ = export_theory ( );}
\]
\[
\text{val _ = print_theory "-";}
\]
\[
\text{end (\text{structure })}
\]
\[
\text{C.2 satList}
\]
\[
\text{(}
\]
\[
\text{(* Definition of satList for conjunctions of ACL formulas *)}
\]
\[
\text{(* Author: Shiu–Kai Chin *)}
\]
\[
\text{(* Date: 24 July 2014 *)}
\]
\[
\text{(* *)}
\]
\[
\text{structure satListScript = struct}
\]
\[
\text{(* interactive mode}
\]
\[
\text{app load "[TypeBase","listTheory","acl_infRules ";]}
\]
\[
\text{(* *)}
\]
\[
\text{open HolKernel boolLib Parse bossLib}
\]
\[
\text{open TypeBase acl_infRules listTheory}
\]
\[
\text{(* ***************}
\]
\[
\text{* create a new theory}
\]
\[
\text{***************)}
\]
\[
\text{val _ = new_theory "satList";}
\]
\[
\text{(* ***********************)}
\]
\[
\text{(* Configurations and policies are represented by lists *)}
\]
\[
\text{(* of formulas in the access–control logic. *)}
\]
\[
\text{(* Previously, for a formula f in the access–control logic, *)}
\]
\[
\text{(* we ultimately interpreted it within the context of a *)}
\]
\[
\text{(* Kripke structure M and partial orders Oi: 'Int po and *)}
\]
\[
\text{(* Os: 'Sec po. This is represented as (M,Oi,O_s) sat f. *)}
\]
\[
\text{(* The natural extension is to interpret a list of formulas *)}
\]
\[
\text{(* [f0;...;fn] as a conjunction: *)}
\]
\[
\text{(* (M,Oi,O_s) sat f0;/...;/ (M,Oi,O_s) sat fn *)}
\]
\[
\text{***********************)}
\]
\[
\text{val _ = set_fixity "satList" (Infixr 540);}
\]
val satList_def = Define
\((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))\)

satList formList = FOLDR
\((\times y. x \land y)\) T
(MAP
\((\{f:('prop,'world,'pName,'Int,'Sec)Form).\)
\((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po)\) sat f)formList)\);

(* Properties of satList *)
(* *******************************)
val satList_nil = TAC_PROOF(
\([],\)
\((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po)) satList []\)

val _ = save_thm("satList_nil",satList_nil)

val satList_conj = TAC_PROOF(
\([],\)
\(!l1 l2 M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po))\) satList l1) /
\(((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po))\) satList l2) =\)

\(((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po))\) satList (l1 ++ l2)\)

val _ = save_thm("satList_conj",satList_conj)

val satList_CONS = TAC_PROOF(
\([],\)
\(!t M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po))\) satList (h:: t)) =\)

\(((M,Oi,Os) sat h) /
\(((M:('prop,'world,'pName,'Int,'Sec)Kripke),\)
\((Oi:'Int po),(Os:'Sec po))\) satList t)\)

val _ = save_thm("satList_CONS",satList_CONS)

val _ = export_theory ()
val _ = print_theory "-";

end (* structure *)
Appendix D

Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files

D.1 OMNI Level

structure OMNIScript = struct

(* INTO Interactive Mode *)
app load ["TypeBase","listTheory","optionTheory","OMNITypeTheory",
"acl_infRules","aclDrulesTheory","aclrulesTheory"];
open TypeBase listTheory optionTheory
open OMNITypeTheory
open acl_infRules aclDrulesTheory aclrulesTheory

val _ = new_theory "OMNI";
(* Define slCommands for OMNI. *)
(* Area 52 *)

val _ = Datatype 'stateRole = Omni'

330
val _ = Datatype `omniCommand = ssmPlanPBComplete
| ssmMoveToORPComplete
| ssmConductORPComplete
| ssmMoveToPBComplete
| ssmConductPBComplete`

val omniCommand_distinct_clauses = distinct_of `omniCommand`
val _ = save_thm("omniCommand_distinct_clauses",
omniCommand_distinct_clauses)

val _ = Datatype `slCommand = OMNI omniCommand`

val omniAuthentication_def = Define
`{(omniAuthentication (Name Omni says prop (cmd:((slCommand command) option)))
 :((slCommand command) option , stateRole , 'd','e)Form) = T) /
 (omniAuthentication _ = F)}`

val omniAuthorization_def = Define
`{(omniAuthorization (Name Omni controls prop (cmd:((slCommand command) option)))
 :((slCommand command) option , stateRole , 'd','e)Form) = T) /
 (omniAuthorization _ = F)}`

This may not be necessary...But, it is interesting. Save for a later time.

(*******************************************************************)
(* Prove that *)
(* Omni says omniCommand ==> omniCommand *)
(*******************************************************************)

set_goal([],
``(Name Omni says prop (cmd:((slCommand command) option))
 :((slCommand command) option , stateRole , 'd','e)Form) ==> prop (cmd:((slCommand command) option))``)

val th1 = ASSUME ``(Name Omni says prop (cmd:((slCommand command) option))
 :((slCommand command) option , stateRole , 'd','e)Form) = T``
val th2 = REWRITE_RULE[omniAuthentication_def]th1

val _ = export_theory();
end

D.2 TopLevel

D.2.1 PBTypeIntegrated Theory: Type Definitions

(*******************************************************************)
(* PBTypeIntegrated *)
(* Author: Lori Pickering *)
(* Date 12 May 2018 *)
(* This theory contains the type definitions for ssmPBIntegrated *)
(*******************************************************************)
structure PBTypeIntegratedScript = struct

331
open HolKernel Parse boolLib bossLib;
open TypeBase OMNITypeTheory

val _ = new_theory "PBTypeIntegrated";

(******************************************************************************)
(* Define types *)
(******************************************************************************)
val _ =
  Datatype `plCommand = crossLD (* Move to MOVE_TO_ORP state *)
    | conductORP
    | moveToPB
    | conductPB
    | completePB
    | incomplete`

val plCommand_distinct_clauses = distinct_of `:plCommand`
val _ = save_thm("plCommand_distinct_clauses", plCommand_distinct_clauses)

val _ =
  Datatype `omniCommand = ssmPlanPBComplete
    | ssmMoveToORPCOMPlete
    | ssmConductORPCOMPlete
    | ssmMoveToPBComplete
    | ssmConductPBComplete
    | invalidOmnCommand`

val omniCommand_distinct_clauses = distinct_of `:omniCommand`
val _ = save_thm("omniCommand_distinct_clauses", omniCommand_distinct_clauses)

val _ =
  Datatype `slCommand = PL plCommand
    | OMNI omniCommand`

val slCommand_distinct_clauses = distinct_of `:slCommand`
val _ = save_thm("slCommand_distinct_clauses", slCommand_distinct_clauses)

val slCommand_one_one = one_one_of `:slCommand`
val _ = save_thm("slCommand_one_one", slCommand_one_one)

val _ =
  Datatype `stateRole = PlatoonLeader | Omni`

val stateRole_distinct_clauses = distinct_of `:stateRole`
val _ = save_thm("stateRole_distinct_clauses", stateRole_distinct_clauses)

val _ =
  Datatype `slState = PLAN_PB
    | MOVE_TO_ORP
    | CONDUCT_ORP
    | MOVE_TO_PB
    | CONDUCT_PB
    | COMPLETE_PB`

val slState_distinct_clauses = distinct_of `:slState`
val _ = save_thm("slState_distinct_clauses", slState_distinct_clauses)
val slOutput = PlanPB
| MoveToORP
| ConductORP
| MoveToPB
| ConductPB
| CompletePB
| unAuthenticated
| unAuthorized

val slOutput_distinct_clauses = distinct_of slOutput
val _ = save_thm("slOutput_distinct_clauses", slOutput_distinct_clauses)

val _ = export_theory();

D.2.2 PBIntegratedDef Theory: Authentication & Authorization Definitions

(* *****************************************************************************)
(* PBIntegratedDefTheory *)
(* Author: Lori Pickering *)
(* Date: 7 May 2018 *)
(* Definitions for ssmPBIntegratedTheory. *)
(* *****************************************************************************)
structure PBIntegratedDefScript = struct

(* ===== Interactive Mode =====
app load ["TypeBase", "listTheory", "optionTheory", "uavUtilities", "OMNITypeTheory", "PBIntegratedDefTheory", "PBTypeIntegratedTheory"];
open TypeBase listTheory optionTheory aclsemanticsTheory aclfoundationTheory OMNITypeTheory PBIntegratedDefTheory PBTypeIntegratedTheory
=== end Interactive Mode ===*)

open HolKernel Parse boolLib bossLib;
onpe TypeBase listTheory optionTheory
donpe uavUtilities
open OMNITypeTheory PBTypeIntegratedTheory

val _ = new_theory "PBIntegratedDef";

(* *****************************************************************************)
(* Helper functions for extracting commands *)
(* *****************************************************************************)
val getPlCom_def = Define ' (getPlCom ([]):((slCommand command)option)list)
  = incomplete:plCommand /
  (getPlCom (SOME (SLc (PL cmd)):(slCommand command)option::xs) = cmd:plCommand) /
  (getPlCom (::_:((xs :(slCommand command)option list))) = (getPlCom xs))'

val secContextNull_def = Define ' secContext (x:((slCommand command)option , stateRole , 'd , 'e)Form list) =
  [(Name Omni) controls prop (SOME (SLc (OMNI omniCommand)))]
val secHelper = Define' (secHelper (cmd:omniCommand) = [(Name Omni) controls prop (SOME (SLc (OMNI (cmd:omniCommand)))))]

val getOmniCommand_def = Define' (getOmniCommand (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = invalidOmniCommand:omniCommand) /
(getOmniCommand ((((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs)
= (cmd:omniCommand)) /
(getOmniCommand ((x:((slCommand command)option , stateRole , 'd,'e)Form)::xs) = (getOmniCommand xs))

val secAuthorization_def = Define' (secAuthorization (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = secHelper (getOmniCommand xs))

val secContext_def = Define' (secContext (PLAN_PB) (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = if ((getOmniCommand xs) = ssmPlanPBComplete:omniCommand) then [(prop (SOME (SLc (OMNI (ssmPlanPBComplete)))))
:( (slCommand command)option , stateRole , 'd,'e)Form) impf (Name PlatoonLeader) controls prop (SOME (SLc (PL crossLD)))
]: (secContext (M O V E _ T O _ O R P ) (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = if (getOmniCommand xs = ssmMoveToORPComplete) then [prop (SOME (SLc (OMNI (ssmMoveToORPComplete)))) impf (Name PlatoonLeader) controls prop (SOME (SLc (PL conductORP)))]
else [prop NONE] ) /
(secContext (C O N D U C T _ O R P ) (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = if (getOmniCommand xs = ssmConductORPComplete) then [prop (SOME (SLc (OMNI (ssmConductORPComplete)))) impf (Name PlatoonLeader) controls prop (SOME (SLc (PL moveToPB)))]
else [prop NONE] ) /
(secContext (MOVE_TO_PB) (xs:((slCommand command)option , stateRole , 'd,'e)Form list) = if (getOmniCommand xs = ssmConductPBComplete) then [prop (SOME (SLc (OMNI (ssmConductPBComplete)))) impf (Name PlatoonLeader) controls prop (SOME (SLc (PL completePB)))]
else [prop NONE] )

(* Area 52 *)

val _ = export_theory();

D.2.3 ssmPlanPBIntegrated Theory: Theorems
This theory aims to integrate the topLevel ssm with the sublevel ssms. It does this by adding a condition to the security context. In particular, this requires that the "COMPLETE" state in the subLevel ssm must precede the transition to the next state at the topLevel. I.e.,

\[ \text{planPBComplete} \Rightarrow \]

PlatoonLeader controls crossLD.

In the ssmPlanPB ssm, the last state is COMPLETE. This is reached when the appropriate authority says complete and the transition is made. I.e.,

\[ \text{planPBComplete} = \Rightarrow \]

PlatoonLeader controls crossLD.

Note that following the ACL, if P says x and P controls x, then x. Therefore, it is not necessary for anyone to say x at the topLevel, because it is already proved at the lower level.

However, indicating that at the topLevel remains something to workout.

structure ssmPBIntegratedScript = struct

(* Interactive Mode *)

val PBNS_def = Define

PBNS PLAN_PB (exec x) = if (getPlCom x) = crossLD then MOVE_TO_ORP else PLAN_PB) /

PBNS MOVE_TO_ORP (exec x) = if (getPlCom x) = conductORP then CONDUCT_ORP else MOVE_TO_ORP) /

PBNS CONDUCT_ORP (exec x) = if (getPlCom x) = moveToPB then MOVE_TO_PB else CONDUCT_ORP) /

PBNS MOVE_TO_PB (exec x) = if (getPlCom x) = conductPB then CONDUCT_PB else MOVE_TO_PB) /

PBNS CONDUCT_PB (exec x) = if (getPlCom x) = completePB then COMPLETE_PB else CONDUCT_PB) /

PBNS (s:slState) (trap _) = s /

PBNS (s:slState) (discard _) = s

val PBOut_def = Define

PBOut PLAN_PB (exec x) = if (getPlCom x) = crossLD then MoveToORP else PlanPB) /

PBOut MOVE_TO_ORP (exec x) = if (getPlCom x) = conductORP then ConductORP else MoveToORP) /

PBOut CONDUCT_ORP (exec x) = if (getPlCom x) = moveToPB then MoveToORP else ConductORP) /

PBOut MOVE_TO_PB (exec x) = if (getPlCom x) = conductPB then ConductPB else MoveToPB) /

PBOut CONDUCT_PB (exec x) = if (getPlCom x) = completePB then CompletePB else ConductPB) /
PBOut (s : slState) (trap _) = unauthorized \/
(PBOut (s : slState) (discard _) = unauthenticated).

(* Define authentication function *)

val inputOK_def = Define
(inputOK (((Name PlatoonLeader) says prop (cmd:((slCommand command)option)))
       :((slCommand command)option, stateRole, 'd, 'e)Form) = T) \/
(inputOK (((Name Omni) says prop (cmd:((slCommand command)option)))
       :((slCommand command)option, stateRole, 'd, 'e)Form) = T) \/
(inputOK _ = F).

(* Prove that commands are rejected unless that are requested by a properly authenticated principal. *)

val inputOK_cmd_reject_lemma = Q. prove ('! cmd. ~(inputOK ((prop (SOME cmd)))'),
   (PROVE_TAC[inputOK_def]))

val _ = save_thm("inputOK_cmd_reject_lemma",
   inputOK_cmd_reject_lemma)

(* Theorem: PlatoonLeader is authorized on crossLD if *)
(* Omni says ssmPlanPBComplete *)

val thPlanPB = ISPECL
[""inputOK:((slCommand command)option, stateRole, 'd, 'e)Form -> bool ",
  "secAuthorization :((slCommand command)option, stateRole, 'd, 'e)Form list ->
      ((slCommand command)option, stateRole, 'd, 'e)Form list ",
  "secContext: (slState) ->
      ((slCommand command)option, stateRole, 'd, 'e)Form list ->
      ((slCommand command)option, stateRole, 'd, 'e)Form list ",
  "[Name Omni] says (prop (SOME (SLc (OMNI ssmPlanPBComplete)))) :
      ((slCommand command)option, stateRole, 'd, 'e)Form;
  (Name PlatoonLeader) says (prop (SOME (SLc (PL crossLD)))) :
      ((slCommand command)option, stateRole, 'd, 'e)Form]",
  "ins:((slCommand command)option, stateRole, 'd, 'e)Form list list ",
  "[PLAN_PB]",
  "outs:slOutput output list trType list"] | TR_exec_cmd_rule

val PlatoonLeader_PLAN_PB_exec_lemma =
TAC_PROOF(
[[], fst(dest_imp(concl thPlanPB))],
REWRITE_TAC[CFGInterpreter_def, secContext_def, secAuthorization_def, secHelper_def,
propCommandList_def, extractPropCommand_def, inputList_def,
getOmniCommand_def,
MAP, extractInput_def, satList_CONS, satList_nil, QSYM satList_conj] THEN
PROVE_TAC[Controls, Modus_Ponens])

val _ = save_thm("PlatoonLeader_PLAN_PB_exec_lemma",
   PlatoonLeader_PLAN_PB_exec_lemma)

val PlatoonLeader_PLAN_PB_exec_justified_lemma =
TAC_PROOF(
[[], snd(dest_imp(concl thPlanPB))],
PROVE_TAC[PlatoonLeader_PLAN_PB_exec_lemma, TR_exec_cmd_rule])

val _ = save_thm("PlatoonLeader_PLAN_PB_exec_justified_lemma",
   PlatoonLeader_PLAN_PB_exec_justified_lemma)

val PlatoonLeader_PLAN_PB_exec_justified_thm =
REWRI TE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def,
extractPropCommand_def, PlatoonLeader_PLAN_PB_exec_lemma]
   PlatoonLeader_PLAN_PB_exec_justified_lemma
val = save_thm("PlatoonLeader_PLAN_PB_trap_lemma",
PlatoonLeader_PLAN_PB_trap_lemma)

(* Theorem: PlatoonLeader is trapped on crossLD if
state = PLAN_PB and
and not Omni says ssmPlanPBComplete
*)

val thPlanPBTrap =
ISPECL
[((slCommand command) option , stateRole , 'd,'e)Form ,
(s: slState) = PLAN_PB ,
((Name Omni) says (prop (SOME (SLc (OMNI omniCommand)))) :((slCommand command) option , stateRole , 'd,'e)Form ,
(PLAN_PB) ,
out:slOutput output list trType list ` ] TR_trap_cmd_rule

val temp2 = fst (dest_imp (concl thPlanPBTrap))

val PlatoonLeader_PLAN_PB_trap_lemma =
TAC_PROOF(
[[]] ,
Term `(omniCommand:omniCommand) = ssmPlanPBComplete) =>
(s:slState) = PLAN_PB =>
"temp2",
DISCH_TAC THEN
DISCH_TAC THEN
ASM_REWRITE_TAC [CFGInterpret_def , secContext_def , secAuthorization_def , secHelper_def ,
propCommandList_def , extractPropCommand_def , inputList_def ,
getOmniCommand_def ,
MAP , extractInput_def , satList_CONS , satList_nil , GSYM satList_conj] THEN
PROVE_TAC [Controls , Modus_Ponen]

val = save_thm("PlatoonLeader_PLAN_PB_trap_lemma",
PlatoonLeader_PLAN_PB_trap_lemma)

val temp3 = snd (dest_imp (concl thPlanPBTrap))

val PlatoonLeader_PLAN_PB_trap_justified_lemma =
TAC_PROOF(
[[]] ,
Term `(omniCommand:omniCommand) = ssmPlanPBComplete) =>
(s:slState) = PLAN_PB =>
"temp3",
DISCH_TAC THEN
DISCH_TAC THEN
PROVE_TAC [PlatoonLeader_PLAN_PB_trap_lemma , TR_trap_cmd_rule]

val = save_thm("PlatoonLeader_PLAN_PB_trap_justified_lemma",
PlatoonLeader_PLAN_PB_trap_justified_lemma)

val PlatoonLeader_PLAN_PB_trap_justified_thm =
REWRITE_RULE [inputList_def , extractInput_def , MAP , propCommandList_def ,
extractPropCommand_def ,
PlatoonLeader_PLAN_PB_trap_lemma ,
PlatoonLeader_PLAN_PB_trap_justified_lemma]

val = save_thm("PlatoonLeader_PLAN_PB_trap_justified_thm",
PlatoonLeader_PLAN_PB_trap_justified_thm)

(* Theorem: PlatoonLeader is not discarded on omniCommand and
*)

337
D.3 Horizontal Slice

D.3.1 ssmPlanPB

D.3.1.1 PlanPBType Theory: Type Definitions

(* PlanPBType contains definitions for datatypes that are used in *)

(* Just playing around with this *)

OK, done fooling around == *)

end
structure PlanPBTypeScript = struct

(* ===== Interactive Mode ===== *)

  app load ["TypeBase"];
  open TypeBase

(* end Interactive Mode *)

open HolKernel Parse boolLib bossLib;
open TypeBase

val _ = new_theory "PlanPBType";

val _ = Datatype `plCommand = receiveMission
  | warno
  | tentativePlan
  | recon
  | report1
  | completePlan
  | opoid
  | supervise
  | report2
  | complete
  | plIncomplete
  | invalidPlCommand`

val plCommand_distinct_clauses = distinct_of `plCommand`
val _ = save_thm("plCommand_distinct_clauses", plCommand_distinct_clauses)

val _ = Datatype `psgCommand = initiateMovement
  | psgIncomplete
  | invalidPsgCommand`

val psgCommand_distinct_clauses = distinct_of `psgCommand`
val _ = save_thm("psgCommand_distinct_clauses", psgCommand_distinct_clauses)

val _ = Datatype `slCommand = PL plCommand
  | PSG psgCommand`

val slCommand_distinct_clauses = distinct_of `slCommand`
val _ = save_thm("slCommand_distinct_clauses", slCommand_distinct_clauses)

val slCommand_one_one = one_one_of `slCommand`
val _ = save_thm("slCommand_one_one", slCommand_one_one)

val _ = Datatype `slState = PLAN_PB
  | RECEIVE_MISSION
  | WARNO
  | TENTATIVE_PLAN
  | INITIATE_MOVEMENT
  | RECON
  | REPORT1
  | COMPLETE_PLAN
  | OPOID
  | SUPERVISE
  | REPORT2
  | COMPLETE`

val slState_distinct_clauses = distinct_of `slState`
val _ = save_thm("slState_distinct_clauses", slState_distinct_clauses)

val _ = Datatype `slOutput = PlanPB
null
:(slCommand command)option, stateRole, 'd','e)Form'

```
(prop (SOME (SLc (PSG initiateMovement)))
 :((slCommand command)option, stateRole, 'd','e)Form'

'(Name PlatoonLeader) controls prop (SOME (SLc (PL report1)))
 :((slCommand command)option, stateRole, 'd','e)Form'``

val PL_notWARNO_Auth_def = Define `'
PL_notWARNO_Auth (cmd:plCommand) =
if [cmd = report1] (* report1 exists WARNO state *)
then
(prop NONE):((slCommand command)option, stateRole, 'd','e)Form
else
((Name PlatoonLeader) says (prop (SOME (SLc (PL cmd))))
 :((slCommand command)option, stateRole, 'd','e)Form::xs)

val getRecon_def = Define `'
(getRecon ([]:((slCommand command)option, stateRole, 'd','e)Form list) = [NONE]) /
(getRecon ((Name PlatoonLeader) says (prop (SOME (SLc (PL recon))))
 :((slCommand command)option, stateRole, 'd','e)Form::xs)

val getTenativePlan_def = Define `'
(getTenativePlan ([]:((slCommand command)option, stateRole, 'd','e)Form list) = [NONE]) /
(getTenativePlan ((Name PlatoonLeader) says (prop (SOME (SLc (PL tentativePlan))))
 :((slCommand command)option, stateRole, 'd','e)Form::xs)

val getReport_def = Define `'
(getReport ([]:((slCommand command)option, stateRole, 'd','e)Form list) = [NONE]) /
(getReport ((Name PlatoonLeader) says (prop (SOME (SLc (PL report1)))))
 :((slCommand command)option, stateRole, 'd','e)Form::xs)

val getInitMove_def = Define `'
(getInitMove ([]:((slCommand command)option, stateRole, 'd','e)Form list) = [NONE]) /
(getInitMove ((Name PlatoonSergeant) says (prop (SOME (SLc (PSG initiateMovement))))
 :((slCommand command)option, stateRole, 'd','e)Form::xs)

val getPlCom_def = Define `'
(getPlCom ([]:((slCommand command)option, stateRole, 'd','e)Form list) = invalidPlCommand)

val getPsgCom_def = Define `'
(getPsgCom ([]:((slCommand command)option, stateRole, 'd','e)Form list) = invalidPsgCommand)
val secContext_def = Define
secContext (x: slState) (y: (slCommand command)option, stateRole, d, e)Form list =
if (x = WARNO) then
  (if (getRecon x = SOME (PL recon))
   (getTenativePlan x = SOME (PL tentativePlan))
   (getReport x = SOME (PL report1))
   (getInitMove x = SOME (PSG initiateMovement)))
then [PL_WARNO_Auth]
else if ((getPlCom x) = invalidPlCommand)
then [(prop NONE)]
else [PL_notWARNO_Auth (getPlCom x)]
(* Back-up copy =*=*=*
* Test theorem for EmitTeX printing. *)
val testTheorem = ASSUME "T=T"
val _ = save_thm("testTheorem", testTheorem)
==== End back-up copy ==== *
val _ = export_theory(); end

D.3.1.3 ssmPlanPB Theory: Theorems

(* ssmPlanPB describes the secure state machine derived from the PLAN_PB *)
(* state at the top-level. This is the first secure state machine to use the *)
(* new ssm parameterizable secure state machine. *)
(* Contributors: Lori Pickering (HOL Implementation) *)
(* Jesse Nathaniel Hall (subject matter) *)

structure ssmPlanPBScript = struct
(* Interactive Mode *)
app load [|"TypeBase", "listTheory", "optionTheory", "listSyntax",
"acl_infrules", "aclDrulesTheory", "aclrulesTheory",
"aclsemanticsTheory", "aclfoundationTheory",
"satListTheory", "ssmTheory", "ssminfrules", "uavUtilities",
"OMNITypeTheory", "PlanPBTtypeTheory", "PlanPBDtypeTheory",
"ssmPlanPBTTheory|];
open TypeBase listTheory optionTheory listSyntax
acl_infrules aclDrulesTheory aclrulesTheory aclsemanticsTheory aclfoundationTheory
satListTheory ssmTheory ssminfrules uavUtilities
OMNITypeTheory PlanPBTtypeTheory PlanPBDtypeTheory
ssmPlanPBTTheory
==== end Interactive Mode ==== *
open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infrules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfrules uavUtilities
open OMNITypeTheory PlanPBTtypeTheory PlanPBDtypeTheory
val _ = new_theory "ssmPlanPB";
(* ************************************************* *)

342
(* Define the next-state & next-output functions for the state machine. *)

val planPBNS_def = Define ' val planPBNS WARNO (exec x) = 
if ((getRecon x = [SOME (SLc (PL recon))]) /
    (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) /
    (getReport x = [SOME (SLc (PL report1))]) /
    (getInitMove x = [SOME (SLc (PSG initiateMovement))]))
then REPORT1 else WARNO) /
(planPBNS PLAN_PB (exec x) = if (getPlCom x = receiveMission)
then RECEIVE_MISSION else PLAN_PB) /
(planPBNS RECEIVE_MISSION (exec x) = if (getPlCom x = warno)
then WARNO else RECEIVE_MISSION) /
(planPBNS REPORT1 (exec x) = if (getPlCom x = completePlan)
then COMPLETE_PLAN else REPORT1) /
(planPBNS COMPLETE_PLAN (exec x) = if (getPlCom x = opoid)
then OPOID else COMPLETE_PLAN) /
(planPBNS OPOID (exec x) = if (getPlCom x = supervise)
then SUPERVISE else OPOID) /
(planPBNS SUPERVISE (exec x) = if (getPlCom x = report2)
then REPORT2 else SUPERVISE) /
(planPBNS REPORT2 (exec x) = if (getPlCom x = complete)
then COMPLETE else REPORT2) /
(planPBNS (s: slState) (trap _) = s) /
(planPBNS (s: slState) (discard _) = s) .

val planPBOut_def = Define ' val planPBOut WARNO (exec x) = 
if ((getRecon x = [SOME (SLc (PL recon))]) /
    (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) /
    (getReport x = [SOME (SLc (PL report1))]) /
    (getInitMove x = [SOME (SLc (PSG initiateMovement))]))
then Report1 else unAuthorized) /
(planPBOut PLAN_PB (exec x) = if (getPlCom x = receiveMission)
then ReceiveMission else unAuthorized) /
(planPBOut RECEIVE_MISSION (exec x) = if (getPlCom x = warno)
then Warno else unAuthorized) /
(planPBOut REPORT1 (exec x) = if (getPlCom x = completePlan)
then CompletePlan else unAuthorized) /
(planPBOut COMPLETE_PLAN (exec x) = if (getPlCom x = opoid)
then Opid else unAuthorized) /
(planPBOut OPOID (exec x) = if (getPlCom x = supervise)
then Supervise else unAuthorized) /
(planPBOut SUPERVISE (exec x) = if (getPlCom x = report2)
then Report2 else unAuthorized) /
(planPBOut REPORT2 (exec x) = if (getPlCom x = complete)
then Complete else unAuthorized) /
(planPBOut (s: slState) (trap _) = unAuthorized) /
(planPBOut (s: slState) (discard _) = unAuthenticated) .

(* inputOK: authentication test function *)

val inputOK_def = Define ' val inputOK ((!(Name PlatoonLeader) says (prop (cmd:((slCommand command)option))) :((slCommand command)option , stateRole , 'd,'e)Form) = T) /
(inputOK ((!(Name PlatoonSergeant) says (prop (cmd:((slCommand command)option)))) :((slCommand command)option , stateRole , 'd,'e)Form) = T) /
(inputOK _ = F) .

(* Any unauthorized command is rejected *)

val inputOK_cmd_reject_lemma = TAC_PROOF( [(] , !cmd. "((inputOK ((prop (SOME cmd)):((slCommand command)option , stateRole , 'd,'e)Form))" ), PROVE_TAC[ inputOK_def ])

343
(* Theorem: PlatoonLeader is authorized on any plCommand *)

(* iff not in WARNO state and *)

(* the plCommand is not report1. *)

(* Helper functions *)

val th1 =
  ISPEC
  (\'inputOK:((slCommand command)option , stateRole , 'd,'e)Form => bool`,
   \'secContextNull :((slCommand command)option , stateRole , 'd,'e)Form list =>
     ((slCommand command)option , stateRole , 'd,'e)Form list`,
   \'secContext: (slState) =>
     ((slCommand command)option , stateRole , 'd,'e)Form list =>
     ((slCommand command)option , stateRole , 'd,'e)Form list`,
   \'(\{(Name PlatoonLeader) says (prop (SOME (SLc (PL plCommand))))\})
     :((slCommand command)option , stateRole , 'd,'e)Form`,
   \'ins:((slCommand command)option , stateRole , 'd,'e)Form list list`,
   \'(s:slState)`
   \'outs:slOutput output list trType list`) TR_exec_cmd_rule

val temp = fst (dest_imp (concl th1))

(* lemma *)

val PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma =
  TAC_PROOF(
    ([], Term `~((s:slState) = WARNO) => ~> (~((plCommand:plCommand) = invalidPlCommand)) => ~> (~((plCommand:plCommand) = report1)) => ~temp`),
    DISCH_TAC THEN
    DISCH_TAC THEN
    DISCH_TAC THEN
    ASM_REWRITE_TAC
      [CFGInterpret_def , secContext_def , secContextNull_def ,
       getRecon_def ; getTenativePlan_def , getReport_def ; getInitMove_def ,
       getPlCom_def , getPsgCom_def , PL_notWARNO_Auth_def ,
       inputList_def , extractInput_def , MAP ,
       propCommandList_def , extractPropCommand_def , satList_CONS ,
       satList_nil , GSYM satList_conj]
    THEN
    PROVE_TAC [Controls , Modus_Ponens])

val _ = save_thm ("PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma",
                    PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma)

(* helper functions *)

val temp2 = snd (dest_imp (concl th1))

(* lemma *)

val PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma =
  TAC_PROOF(
    ([], Term `~((s:slState) = WARNO) => ~> (~((plCommand:plCommand) = invalidPlCommand)) => ~> (~((plCommand:plCommand) = report1)) => ~temp2`),
    DISCH_TAC THEN
    DISCH_TAC THEN
    DISCH_TAC THEN
    PROVE_TAC
      [PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma ,
       TR_exec_cmd_rule]

val _ = save_thm ("PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma",
                    PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma)

(* Main theorem *)

val PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm =
  REWRITE_RULE
    [propCommandList_def , inputList_def , extractPropCommand_def ,
     extractInput_def , MAP] PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma
PlatoonLeader is authorized on any report1 if this is the WARNO state and PlatoonLeader says recon /\ PlatoonLeader says tentativePlan /\ PlatoonSergeant says initiateMovement /\ PlatoonLeader says report1

\[
\begin{align*}
&\text{PL}_{\text{WARNO}}\text{Auth} \\
&\text{propCommandList}_\text{def}, \text{inputList}_\text{def}, \text{extractPropCommand}_\text{def}, \\
&\text{extractInput}_\text{def}, \text{MAP}, \text{getRecog}_\text{def}, \text{getTentativePlan}_\text{def}, \text{getReport}_\text{def}, \text{getInitMove}_\text{def}, \\
&\text{getPlCom}_\text{def}, \text{getPsgCom}_\text{def}, \text{PL}_{\text{WARNO}}\text{Auth}_\text{def}
\end{align*}
\]
Theorem: PlatoonLeader is not discarded any psgCommand.

Note that this is just meant to demonstrate the authenticated inputs are not discarded. Instead, they should be trapped. This is because of how the inputOK (authentication) was defined. Note, this proof would also be valid for PlatoonLeader on any plCommand. It is not necessary to prove this.

Should we put this GENL in the TR_discard_cmd_rule?

val thld =

    GENL
      ["(elementTest : (command option, 'principal', 'd', 'e') Form -> bool)",
       "(context : (command option, 'principal', 'd', 'e') Form list ->
          (command option, 'principal', 'd', 'e') Form list)"]

    (stateInterp :
      '"state"
       ->

       "(command option, 'principal', 'd', 'e') Form list ->

       "(command option, 'principal', 'd', 'e') Form list"

      "(ins : (command option, 'principal', 'd', 'e') Form list list)",

      "(s : 'state')",

      "(outs : 'output list")",

      "(NS : 'state -> 'command option list trType -> 'state")",

      "(Out : 'state -> 'command option list trType -> 'output")",

      "(M : (command option, 'b', 'principal', 'd', 'e') Kripke)"

     "(Oi : 'd po)",

     "(Os : 'e po)"
    ]

```
TR_discard_cmd_rule
```

val th2d =

ISPECL

["inputOK: ((slCommand command) option, stateRole, 'd', 'e') Form -> bool",

"secContextNull: ((slCommand command) option, stateRole, 'd', 'e') Form list ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list",

"secContext: (s: slState) ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list",

"(Name PlatoonLeader) says (prop (SOME (SLc (PSG psgCommand))))",

"(s: slState)"

"outs: slOutput output list trType list") th1d

val th3d =

LIST_BETA_CONV (Term "\(\forall p q. p \land q \rightarrow T \land ((\forall p q. p \land q) \rightarrow T)\")

val th3d2 =

LIST_BETA_CONV (Term "\(\forall p q. p \land q \rightarrow T\)

val PlatoonLeader_psgCommand_notdiscard_thm =

REWRITE_RULE

[inputList_def, extractInput_def, authenticationTest_def, MAP, inputOK_def, FOLDR, th3d, th3d2] th2d

val _ = save_thm("PlatoonLeader_psgCommand_notDiscard_thm",

   PlatoonLeader_psgCommand_notDiscard_thm)

(* Theorem: PlatoonLeader is trapped on any psgCommand. *)

val thlt =

ISPECL

["inputOK: ((slCommand command) option, stateRole, 'd', 'e') Form -> bool",

"secContextNull: ((slCommand command) option, stateRole, 'd', 'e') Form list ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list",

"secContext: (s: slState) ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list ->
   ((slCommand command) option, stateRole, 'd', 'e') Form list",

"(Name PlatoonLeader) says (prop (SOME (SLc (PSG psgCommand))))",

"(s: slState)"

"outs: slOutput output list trType list")

```
TR_trap_cmd_rule
```

346
val PlatoonLeader_trap_psgCommand_lemma = 
TAC_PROOF(([] ,
  fst(dest_imp(concl th1t))) ,
ASM_REWRITE_TAC
[CFGInterpret_def , inputOK_def , secContext_def , secContextNull_def]
THEN
ASM_REWRITE_TAC
[getRecon_def , getTentativePlan_def , getReport_def , getInitMove_def ,
getPlCom_def , satList_CONS , satList_nil , GSYM satList_conj]
THEN
ASM_REWRITE_TAC
[NOT_NONE_SOME , NOT_SOME_NONE , SOME_11 , list_11 ,
plCommand_distinct_clauses , psgCommand_distinct_clauses ,
GSYM slCommand_distinct_clauses ,
GSYM plCommand_distinct_clauses ,
GSYM psgCommand_distinct_clauses]
THEN
PROVE_TAC[satList_CONS , satList_nil]
)
val _ = save_thm("PlatoonLeader_trap_psgCommand_lemma",
  PlatoonLeader_trap_psgCommand_lemma)

val PlatoonLeader_trap_psgCommand_justified_lemma =
TAC_PROOF(([] ,
  snd(dest_imp(concl th1t))) ,
PROVE_TAC
[PlatoonLeader_trap_psgCommand_lemma , TR_trap_cmd_rule])
val _ = save_thm("PlatoonLeader_trap_psgCommand_justified_lemma",
  PlatoonLeader_trap_psgCommand_justified_lemma)

val PlatoonLeader_trap_psgCommand_justified_thm =
REWRITE_RULE
[propCommandList_def , inputList_def , extractPropCommand_def ,
extractInput_def , MAP]
PlatoonLeader_trap_psgCommand_justified_lemma
val _ = save_thm("PlatoonLeader_trap_psgCommand_justified_lemma",
  PlatoonLeader_trap_psgCommand_justified_lemma)

(* Theorem: PlatoonSergeant is trapped on any plCommand. *)

(* Theorem: *)

val th1tt =
ISPECL ["inputOK:((slCommand command) option , stateRole , 'd','e')Form -> bool" ,
  secContextNull :((slCommand command) option , stateRole , 'd','e')Form list ->
  ((slCommand command) option , stateRole , 'd','e')Form list" ,
  secContext: (slState) ->
  ((slCommand command) option , stateRole , 'd','e')Form list ->
  ((slCommand command) option , stateRole , 'd','e')Form list" ,
  "[(Name PlatoonSergeant) says (prop SOME (SLc (PL plCommand)))]
  :((slCommand command) option , stateRole , 'd','e')Form list" ,
  "ins:((slCommand command) option , stateRole , 'd','e')Form list list" ,
  "(s:slState)" ,
  "outs:slOutput output list trType list"]
TR_trap_cmd_rule

val PlatoonSergeant_trap_plCommand_lemma =
TAC_PROOF(([] ,
  fst(dest_imp(concl th1tt))) ,
ASM_REWRITE_TAC
[CFGInterpret_def , inputOK_def , secContext_def , secContextNull_def]
THEN
ASM_REWRITE_TAC
[getRecon_def , getTentativePlan_def , getReport_def , getInitMove_def ,
getPlCom_def , satList_CONS , satList_nil , GSYM satList_conj]

347
THEN
ASM_REWRITE_TAC
\[\{ NOT\_NONE\_SOME, NOT\_SOME\_NONE, SOME\_11,\]
slCommand\_one\_one, slCommand\_distinct\_clauses,
plCommand\_distinct\_clauses, psgCommand\_distinct\_clauses,
\text{GSYM} slCommand\_distinct\_clauses,
\text{GSYM} plCommand\_distinct\_clauses,
\text{GSYM} psgCommand\_distinct\_clauses\]  
THEN
PROVE_TAC \[\{ satList\_CONS, satList\_nil\}\]

val _ = save_thm("PlatoonSergeant\_trap\_plCommand\_lemma",
PlatoonSergeant\_trap\_plCommand\_lemma)

val PlatoonSergeant\_trap\_plCommand\_justified\_lemma =
TAC_PROOF(
\[(||,\]
snd(dest_imp(concl th1tt))),
PROVE_TAC
\{PlatoonSergeant\_trap\_plCommand\_lemma, TR\_trap\_cmd\_rule\})

val _ = save_thm("PlatoonSergeant\_trap\_plCommand\_justified\_lemma",
PlatoonSergeant\_trap\_plCommand\_justified\_lemma)

val PlatoonSergeant\_trap\_plCommand\_justified\_thm =
REWRITE_RULE
\{propCommandList\_def, inputList\_def, extractPropCommand\_def,
extractInput\_def, MAP\}
PlatoonSergeant\_trap\_plCommand\_justified\_lemma

val _ = save_thm("PlatoonSergeant\_trap\_plCommand\_justified\_thm",
PlatoonSergeant\_trap\_plCommand\_justified\_thm)

(* === Start testing here ===*)

----- End Testing Here ----- *)
val _ = export_theory();

end

D.3.2 ssmMoveToORP

D.3.2.1 MoveToORPType Theory: Type Definitions & Authentication &
Authorization Definitions

(* *******************************************************************************)
(* MoveToORPType contains definitions for datatypes that are used in         *)
(* the MoveToORP state machine.                                              *)
(* Contributors:                                                            *)
(* Lori Pickering (HOL implementation),                                     *)
(* Jesse Hall (content expert).                                             *)
(* Prof. Shiu-Kai Chin (Principal Investigator).                           *)
(* Date created: 19 June 2017                                              *)
(* *******************************************************************************)
structure MoveToORPScript = struct

(* ===== Interactive Mode =====*)
app load ["TypeBase"]
open TypeBase
===== end Interactive Mode ===== *)

open HolKernel Parse boolLib bossLib;
open TypeBase
val _ = new_theory "MoveToORPType";
val `slCommand =\n| \text{pltForm}\n| \text{pltMove}\n| \text{pltSecureHalt}\n| \text{complete}\n| \text{incomplete}'

val `slCommand_distinct_clauses = distinct_of `:\text{slCommand}\`
val _ = save_thm('\text{slCommand_distinct_clauses}',\text{slCommand_distinct_clauses})

val `slState =\n| \text{MOVE\_TO\_ORP}\n| \text{PLT\_FORM}\n| \text{PLT\_MOVE}\n| \text{PLT\_SECURE\_HALT}\n| \text{COMPLETE}'

val `slState_distinct_clauses = distinct_of `:\text{slState}\`
val _ = save_thm('\text{slState_distinct_clauses}',\text{slState_distinct_clauses})

val `slOutput =\n| \text{MoveToORP}\n| \text{PLTForm}\n| \text{PLTMove}\n| \text{PLTSecureHalt}\n| \text{Complete}\n| \text{unAuthorized}\n| \text{unAuthenticated}'

val `slOutput_distinct_clauses = distinct_of `:\text{slOutput}\`
val _ = save_thm('\text{slOutput_distinct_clauses}',\text{slOutput_distinct_clauses})

val `stateRole =\n| \text{PlatoonLeader}'

val _ = export_theory();

\end

\D.3.2.2 ssmMoveToORP Theory: Theorems

\textbf{structure ssmMoveToORPScript = struct}

\textbf{app load \{"TypeBase", "listTheory","optionTheory",\n| "acl_infRules","aclDrulesTheory","aclrulesTheory",\n| "satListTheory","ssm11Theory","ssminfRules",\n| "OMNITypeTheory", "MoveToORPTypeTheory"\};

open TypeBase listTheory optionTheory
acl_infRules aclDrulesTheory aclrulesTheory
satListTheory ssm11Theory ssminfRules
OMNITypeTheory MoveToORPTypeTheory

ssmMoveToORPTheory
\textbf{=== end Interactive Mode === \*)
open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open aclInfRules aclDrulesTheory aclrulesTheory
open satListTheory ssm11Theory ssmInfRules
open OMNITypeTheory MoveToORPTypeTheory

val _ = new_theory "ssmMoveToORP";

(* Define the next state function for the state machine. *)
val moveToORPNS_def =
  Define
    moveToORPNS MOVE_TO_ORP (exec (SLc pltForm)) = PLT_FORM /
    moveToORPNS MOVE_TO_ORP (exec (SLc incomplete)) = MOVE_TO_ORP /
    moveToORPNS PLT_FORM (exec (SLc pltMove)) = PLT_MOVE /
    moveToORPNS PLT_MOVE (exec (SLc incomplete)) = PLT_MOVE /
    moveToORPNS PLT_SECURE_HALT (exec (SLc complete)) = COMPLETE /
    moveToORPNS PLT_SECURE_HALT (exec (SLc incomplete)) = PLT_SECURE_HALT /

(* trapping *)
moveToORPNS (s : slState) (trap (SLc (cmd : slCommand))) = s /

(* discarding *)
moveToORPNS (s : slState) (discard (SLc (cmd : slCommand))) = s'

(* Define next-output function for the state machine *)
val moveToORPOut_def =
  Define
    moveToORPOut MOVE_TO_ORP (exec (SLc pltForm)) = PLTForm /
    moveToORPOut MOVE_TO_ORP (exec (SLc incomplete)) = MoveToORP /
    moveToORPOut PLT_FORM (exec (SLc pltMove)) = PLTMove /
    moveToORPOut PLT_MOVE (exec (SLc incomplete)) = PLTMove /
    moveToORPOut PLT_SECURE_HALT (exec (SLc complete)) = Complete /
    moveToORPOut PLT_SECURE_HALT (exec (SLc incomplete)) = PLTSecureHalt /

(* trapping *)
moveToORPOut (s : slState) (trap (SLc (cmd : slCommand))) = unAuthorized /

(* discarding *)
moveToORPOut (s : slState) (discard (SLc (cmd : slCommand))) = unAuthenticated'

(* Input Authentication *)
val authTestMoveToORP_def =
  Define
    'authTestMoveToORP
      ((Name PlatoonLeader) says (prop (cmd : (slCommand command) order)
        : ((slCommand command) order, stateRole, 'd, 'e)Form) = T) /
      (authTestMoveToORP _ = F)'

(* State Interpretation: this is the trivial assumption TT, as the machine *)
(* state has no influence on access privileges"—Prof. Chin, SM0Script.sml *)
val ssmMoveToORPStateInterp_def =
  Define
    'ssmMoveToORPStateInterp (state : slState state) =
      (TT : ((slCommand command) order, stateRole, 'd, 'e)Form)'

350
A theorem showing commands without a principal are rejected. --Prof

Chin, SM0Script.sml

val authTestMoveToORP_cmd_reject_lemma =

TAC_PROOF(
    ([],
     "authTestMoveToORP
       (prop (SOME cmd)):((slCommand command)order, stateRole, 'd', 'e')Form)`,
     PROVE_TAC[authTestMoveToORP_def])

val _ = save_thm("authTestMoveToORP_cmd_reject_lemma",
                  authTestMoveToORP_cmd_reject_lemma)

(* securityContext definition: PlatoonLeader authorized on any slCommand
   (defined in PBTTypeScript.sml)

   val secContextMoveToORP_def =
   Define
     secContextMoveToORP cmd =
       (((Name PlatoonLeader) controls
           (prop (SOME (SLc slCommand)))))::ins)
  CFGInterpret ((M:((slCommand command)order, stateRole, 'd', 'e')Kripke),Oi,Os)
   (secContextMoveToORP slCommand)
   (((Name PlatoonLeader) says (prop (SOME (SLc slCommand))))::ins)
   ((s:slState state)
     (outs:(slOutput output) list) ) =>
   ((M,Oi,Os) sat (prop (SOME (SLc slCommand))))`,
   REWRITE_TAC[CFGInterpret_def,secContextMoveToORP_def,ssmMoveToORPStateInterp_def,
                 satList_CONS,satList_nil,sat_TT] THEN
   PROVE_TAC[Controls]
)

val _ = save_thm("PlatoonLeader_slCommand_lemma",
                  PlatoonLeader_slCommand_lemma)

(* exec slCommand occurs if and only if PlatoonLeaders's command is
   authenticated and authorized

   val PlatoonLeader_exec_slCommand_justified_thm =
   let
     val th1 =
       ISPECL
         [''authTestMoveToORP:((|(slCommand command)order, stateRole, 'd', 'e')Form => bool''),
          '(secContextMoveToORP slCommand):(|(slCommand command)order, stateRole, 'd', 'e')Form list'','
          'ssmMoveToORPStateInterp:(|(slState state)->
            (|(slCommand command)order, stateRole, 'd', 'e')Form'','
          'Name PlatoonLeader''',''SLc slCommand:(|slCommand command)'','
          'ins:((|slCommand command)order, stateRole, 'd', 'e')Form list'','
          's:|(slState state)'',''outs:|(slOutput output) list'']
       TR_exeexec_cmd_rule
     in
       TAC_PROOF(
         [],
         "(|ns:|(slState state) -> (|(slCommand command)trType -> (slState state))
          (out:|(slState state) -> (|(slCommand command)trType -> (slOutput output)))
          (m:|(slCommand command)order, 'b', stateRole, 'd', 'e')Kripke)
          (oi:'d'p)
          (os:'e'p).
          TR (M,Oi,Os) (exec (SLc slCommand):(|(slCommand command)trType))
       CFG
         (authTestMoveToORP
          :(|(slCommand command)order, stateRole, 'd', 'e')Form => bool)
         (ssmMoveToORPStateInterp :|(slState state)"

351
D.3.3 ssmConductORP

D.3.3.1 ConductORPType Theory: Type Definitions

(* *******************************************************)
ConductORPType contains definitions for datatypes that are used in the conductORP state machine ssmConductORP.

Contributors:
- Lori Pickering (HOL implementation),
- Jesse Hall (content expert),
- Prof. Shiu-Kai Chin (Principal Investigator).

Date created: 16 June 2017

structure ConductORPTypeScript = struct
  (** Interactive Mode ====
app load ["TypeBase"];
open TypeBase
  *** end Interactive Mode === *)

open HolKernel Parse boolLib bossLib;
open TypeBase

val _ = new_theory "ConductORPType";
  (** slcommand, slState, slOutput, and stateRole *)
val _ = save_thm("slcommand_distinct_clauses",slcommand_distinct_clauses)
val _ = Datatype `plCommand = secure
  | withdraw
  | complete
  | p1Incomplete`
val plCommand_distinct_clauses = distinct_of `:plCommand`;
val _ = save_thm("plCommand_distinct_clauses",plCommand_distinct_clauses)
val _ = Datatype `psgCommand = actionsIn
  | psgIncomplete`
val psgCommand_distinct_clauses = distinct_of `:psgCommand`;
val _ = save_thm("psgCommand_distinct_clauses",psgCommand_distinct_clauses)
val _ = Datatype `omniCommand = ssmSecureComplete
  | ssmActionsIncomplete
  | ssmWithdrawComplete
  | invalidOmniCommand`
val omniCommand_distinct_clauses = distinct_of `:omniCommand`;
val _ = save_thm("omniCommand_distinct_clauses",omniCommand_distinct_clauses)
val _ = Datatype `slCommand = PL plCommand
  | PSG psgCommand
  | OMNI omniCommand`
val slCommand_distinct_clauses = distinct_of `:slCommand`;
val _ = save_thm("slCommand_distinct_clauses",slCommand_distinct_clauses)
val slCommand_one_one = one_one_of `:slCommand`;
val _ = save_thm("slCommand_one_one",slCommand_one_one)

(********states *******************)
val _ = Datatype `slState = CONDUCT ORP
  | SECURE
  | ACTIONS IN
  | WITHDRAW
  | COMPLETE`
val slState_distinct_clauses = distinct_of `:slState`;
val _ = save_thm("slState_distinct_clauses",slState_distinct_clauses)

(********output *******************)
val _ =
Datatype `slOutput = ConductORP
| Secure
| ActionsIn
| Withdraw
| Complete
| unAuthenticated
| unAuthorized`

val slOutput_distinct_clauses = distinct_of `:slOutput``
val _ = save_thm("slOutput_distinct_clauses",slOutput_distinct_clauses)

(*****************************************************************************principals *****************************************************************************)
val _ =
Datatype `stateRole = PlatoonLeader
| PlatoonSergeant
| Omni`

val sIRole_distinct_clauses = distinct_of `:stateRole``
val _ = save_thm("sIRole_distinct_clauses",sIRole_distinct_clauses)

val _ = export_theory();
end

D.3.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

(*****************************************************************************)
(* ConductORPDef *)
(* Author : Lori Pickering *)
(* Date : 11 August 2018 *)
(* Definitions for ssmConductORPDef *)
(*****************************************************************************)

structure ConductORPDefScript = struct

(* ====== Interactive Mode ====== *)
app load ["TypeBase","listTheory","optionTheory",
  "acl_infRules","aclrulesTheory","aclrulesTheory",
  "satListTheory","ssmTheory","ssminfRules",
  "OMNITypeTheory","ConductORPTypeTheory";]

"ssmConductORPTheory");
open TypeBase listTheory optionTheory
acl_infRules aclrulesTheory aclrulesTheory
satListTheory ssmTheory ssminfRules
OMNITypeTheory ConductORPTypeTheory
ssmConductORPTypeTheory
==== end Interactive Mode ==== *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules
open OMNITypeTheory ConductORPTypeTheory

val _ = new_theory "ConductORPDef";

(*****************************************************************************)
(* Helper functions for extracting commands *)
(*****************************************************************************)

val getPICom_def =
  Define:
  (getPICom [ ]:(slCommand command)option list) = pIncomplete:plCommand) /
  (getPICom SOME (SLc (PL cmd)):slCommand command)option::xs) = cmd:plCommand) /
  (getPICom _::(xs :(slCommand command)option list))
val getPsGCom_def = Define
  (getPsGCom ([]):((slCommand command)option) list
    = psgIncomplete:psgCommand) /
  (getPsGCom (SOME (SLc (PSG cmd)):(slCommand command)option::xs)
    = cmd:psgCommand) /
  (getPsGCom (_::(xs :(slCommand command)option list))
    = (getPsGCom xs))

(* security context, non state-dependent. *)

val secHelper = Define
  (secHelper (cmd:omniCommand) =
    [(Name Omni) controls prop (SOME (SLc (OMNI cmd)))])

val getOmniCommand_def = Define
  (getOmniCommand ([]:((slCommand command)option, stateRole,
                'd','e)Form list)
    = invalidOmniCommand:omniCommand) /
  (getOmniCommand (((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs)
    = (cmd:omniCommand)) /
  (getOmniCommand ((x:((slCommand command)option, stateRole,
                'd','e)Form)::xs)
    = (getOmniCommand xs))

val secAuthorization_def = Define
  (secAuthorization (xs:((slCommand command)option, stateRole,
                    'd','e)Form list)
    = secHelper (getOmniCommand xs))

(* security context, state-dependent. *)

val secContext_def = Define
  (secContext (CONDUCT ORP) (xs:((slCommand command)option, stateRole,
                   'd','e)Form list) =
    [(Name PlatoonLeader) controls prop (SOME (SLc (PL secure)))
       :((slCommand command)option, stateRole,
       'd','e)Form]) /
  (secContext (SECURE) (xs:((slCommand command)option, stateRole,
                   'd','e)Form list) =
    if ((getOmniCommand xs) = ssmSecureComplete:omniCommand)
    then [(prop (SOME (SLc (OMNI ssmSecureComplete)))
          :((slCommand command)option, stateRole,
          'd','e)Form) impf
           (Name PlatoonSergeant) controls prop (SOME (SLc (PSG actionsIn)))
          :((slCommand command)option, stateRole,
          'd','e)Form)
    else [prop NONE]) /
  (secContext (ACTIONS IN) (xs:((slCommand command)option, stateRole,
                   'd','e)Form list) =
    if ((getOmniCommand xs) = ssmActionsInComplete)
    then [(prop (SOME (SLc (OMNI ssmActionsInComplete)))
          :((slCommand command)option, stateRole,
          'd','e)Form) impf
           (Name PlatoonLeader) controls prop (SOME (SLc (PL withdraw)))
           :((slCommand command)option, stateRole,
           'd','e)Form)
    else [prop NONE]) /
  (secContext (WITHDRAW) (xs:((slCommand command)option, stateRole,
                   'd','e)Form list) =
    if ((getOmniCommand xs) = ssmWithdrawComplete)
    then [(prop (SOME (SLc (OMNI ssmWithdrawComplete)))
          :((slCommand command)option, stateRole,
          'd','e)Form) impf
           (Name PlatoonLeader) controls prop (SOME (SLc (PL complete)))
           :((slCommand command)option, stateRole,
           'd','e)Form)
    else [prop NONE])

val _ = export_theory();

D.3.3.3 ssmConductORP Theory: Theorems

(******************************************************************************)
(* ssmConductORP defines the ConductORP sub-level state machine for the  *)
(* patrol base.                                                       *)
(* some have sub-sub-level state machines. These are implemented in separate *)
(* theories.                                                *)
(* Author: Lori Pickering in collaboration with Jesse Nathaniel Hall *)
(* Date: 16 July 2017                                                   *)
(******************************************************************************)

structure ssmConductORPScript = struct

(* Interactive Mode *)

app load ["TypeBase", "listTheory", "optionTheory",
"acl_infRules", "aclDrulesTheory", "aclrulesTheory",
"satListTheory", "ssmTheory", "ssminfRules",
"OMNITypeTheory",
"ConductORPTypeTheory", "ConductORPDefTheory",
"ssmConductORPTheory"];

open TypeBase listTheory optionTheory
open acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules
open OMNITypeTheory ConductORPTypeTheory ConductORPDefTheory

val _ = new_theory "ssmConductORP";

(* Define the next state function for the state machine. *)

val conductORPNs_def = Define

(val conductORPNs CONDUCT_ORP (exec x) =
   if (getPicom x) = secure then SECURE else CONDUCT_ORP) /
(val conductORPNs SECURE (exec x) =
   if (getPsgcom x) = actionsIn then ACTIONS_IN else SECURE) /
(val conductORPNs ACTIONS_IN (exec x) =
   if (getPicom x) = withdraw then WITHDRAW else ACTIONS_IN) /
(val conductORPNs WITHDRAW (exec x) =
   if (getPicom x) = complete then COMPLETE else WITHDRAW) /
(* trapping *)
(val conductORPNs (s:slState) (trap x) =)
(* discarding *)
(val conductORPNs (s:slState) (discard x) =)

(* Define next-output function for the state machine *)

val conductORPOut_def = Define

(val conductORPOut CONDUCT_ORP (exec x) =
   if (getPicom x) = secure then Secure else ConductORP) /
(val conductORPOut SECURE (exec x) =
   if (getPsgcom x) = actionsIn then ActionsIn else Secure) /
(val conductORPOut ACTIONS_IN (exec x) =
   if (getPicom x) = withdraw then Withdraw else ActionsIn) /
(val conductORPOut WITHDRAW (exec x) =
   if (getPicom x) = complete then Complete else Withdraw) /
(* trapping *)
(val conductORPOut (s:slState) (trap x) = unAuthorized) /
(* discarding *)

356
(conductORPOut (s:slState) (discard x) = unAuthenticated)

(* Input Authentication *)

val inputOK_def = Define
  `\((\text{Name PlatoonLeader}) \text{ says } (\text{prop} (\text{cmd:(slCommand command)option}))
   :((\text{slCommand command)option ,stateRole ,\text{d}',e)Form) = T) \land
  (\text{inputOK}
   ((\text{Name PlatoonSergeant}) \text{ says } (\text{prop} (\text{cmd:(slCommand command)option}))
   :((\text{slCommand command)option ,stateRole ,\text{d}',e)Form) = T) \land
  (\text{inputOK}
   ((\text{Name Omni}) \text{ says } (\text{prop} (\text{cmd:(slCommand command)option}))
   :((\text{slCommand command)option ,stateRole ,\text{d}',e)Form) = T) \land
  (\text{inputOK} = F)`

(* "A theorem showing commands without a principal are rejected." --- Prof *)

val inputOK_cmd_reject_lemma = TAC_PROOF(
  [[] ,
  \((!\text{cmd}. \neg (\text{inputOK}
    ((\text{prop} (\text{SOME cmd})):((\text{slCommand command)option ,stateRole ,\text{d}',e)Form})) ))
  \text{PROVE_TAC [inputOK_def]}
)

val _ = save_thm("inputOK_cmd_reject_lemma*",
  inputOK_cmd_reject_lemma)

(* PlatoonLeader is justified on secure. *)

val th1 = ISPECL
  [[] ,
  \("!\text{cmd}. \neg (\text{inputOK}
    ((\text{prop} (\text{SOME cmd})):((\text{slCommand command)option ,stateRole ,\text{d}',e)Form}))
    \text{PROVE_TAC[inputOK_def]})"
  \text{PROVE_TAC[CFGInterpret_def, secContext_def, secAuthorization_def,
    getOmniCommand_def, inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS,
    satList_nil, GSYM satList_conj] THEN
    PROVE_TAC[Controls]})

val _ = save_thm("PlatoonLeader\_CONDUCT\_ORP\_exec\_secure\_lemma",
  PlatoonLeader\_CONDUCT\_ORP\_exec\_secure\_lemma)

val PlatoonLeader\_CONDUCT\_ORP\_exec\_secure\_justified_thm = TAC_PROOF(
  [[] ,
  snd(dest_imp(concl th1))],
  PROVE_TAC[PlatoonLeader\_CONDUCT\_ORP\_exec\_secure\_lemma, TR\_exec\_cmd\_rule])
val _ = save_thm("PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm",
    PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm)

val PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm =
    REWRITE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm]

val _ = save_thm("PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm",
    PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm)

(* PlatoonLeader is justified on actionsIn if
(* if Omni says ssmSecureComplete.

(* PlatoonSergeant is justified on actionsIn if
(* if Omni says ssmSecureComplete.

val th1 = ISPECL[
    (\inputOK : ((slCommand command) option , stateRole , 'd','e)Form => bool' ,
    'secAuthorization :((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    'secContext : (slState) =>
    ((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    '\[(Name Omni) says (prop (SOME (SLc (OMNI ssmSecureComplete))))\]
    (Name PlatoonSergeant) says (prop (SOME (SLc (PSG actionsIn))))\]
    (\ins :((slCommand command)option , stateRole , 'd','e)Form list list \',
    '\(SECURE)\',
    '\outs :slOutput output list trType list \') TR_exec_cmd_rule

val PlatoonSergeant_SECURE_exec_lemma =
    TAC_PROOF(\
    [|\tinputOK:((slCommand command) option , stateRole , 'd','e)Form => bool\',
    \tsecAuthorization :((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    \tsecContext : (slState) =>
    ((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    \t\[(Name Omni) says (prop (SOME (SLc (OMNI ssmSecureComplete))))\]
    (Name PlatoonSergeant) says (prop (SOME (SLc (PSG actionsIn))))\]
    \tsnd(dest_imp(concl th1)))],
    PROVE_TAC[PlatoonSergeant_SECURE_exec_lemma, TR_exec_cmd_rule])

val _ = save_thm("PlatoonSergeant_SECURE_exec_justified_lemma",
    PlatoonSergeant_SECURE_exec_justified_lemma)

val PlatoonSergeant_SECURE_exec_justified_lemma =
    TAC_PROOF(\
    [|\tinputOK:((slCommand command) option , stateRole , 'd','e)Form => bool\',
    \tsecAuthorization :((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    \tsecContext : (slState) =>
    ((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    \t\[(Name Omni) says (prop (SOME (SLc (OMNI ssmSecureComplete))))\]
    (Name PlatoonSergeant) says (prop (SOME (SLc (PSG actionsIn))))\]
    \tsnd(dest_imp(concl th1)))],
    PROVE_TAC[PlatoonSergeant_SECURE_exec_justified_lemma, TR_exec_cmd_rule])

val _ = save_thm("PlatoonSergeant_SECURE_exec_justified_thm",
    PlatoonSergeant_SECURE_exec_justified_thm)

(* PlatoonLeader is justified on withdraw if
(* if Omni says ssmActionsInComplete.

(* PlatoonLeader is justified on withdraw if
(* if Omni says ssmActionsInComplete.

val th1 = ISPECL[
    (\inputOK:((slCommand command) option , stateRole , 'd','e)Form => bool\',
    'secAuthorization :((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    'secContext : (slState) =>
    ((slCommand command)option , stateRole , 'd','e)Form list =>
    ((slCommand command)option , stateRole , 'd','e)Form list\',
    'INS:
    'OUTS:slOutput output list trType list \'] TR_exec_cmd_rule
PlatoonLeader is trapped on withdraw if
if not Omni says ssmActionsInComplete.
PROVE_TAC[Controls, Modus_Ponens])

val _ = save_thm("PlatoonLeader_ACTIONS_IN_trap_lemma", PlatoonLeader_ACTIONS_IN_trap_lemma)

val temp3 = snd(dest_imp(concl thTrap))

val PlatoonLeader_ACTIONS_IN_trap_justified_lemma = TAC_PROOF([],
Term\((\{\text{omniCommand:\text{omniCommand}} = \text{ssmActionsInComplete}\}) \implies\n((s:\text{slState}) = \text{ACTIONS_IN}) \implies\n\text{temp3})\),
DISCH_TAC THEN DISCH_TAC THEN PROVE_TAC[PlatoonLeader_ACTIONS_IN_trap_lemma, TR_trap_cmd_rule])

val _ = save_thm("PlatoonLeader_ACTIONS_IN_trap_justified_lemma", PlatoonLeader_ACTIONS_IN_trap_justified_lemma)

val PlatoonLeader_ACTIONS_IN_trap_justified_thm = REWRITE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, PlatoonLeader_ACTIONS_IN_trap_lemma] PlatoonLeader_ACTIONS_IN_trap_justified_lemma

val _ = save_thm("PlatoonLeader_ACTIONS_IN_trap_justified_thm", PlatoonLeader_ACTIONS_IN_trap_justified_thm)

(* ===== Interactive Mode =====

====== Interactive Mode ===== *)

val _ = export_theory();
end

D.3.4 ssmMoveToPB

D.3.4.1 MoveToPBType Theory: Type Definitions

(*******************************************************************)
(* projectTypesScript ; sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project's *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-oneness theorems. *)
(* Project Description: ssmMoveToPB. *)
(*******************************************************************)
structure projectTypesScript = struct

open HolKernel Parse boolLib bossLib bossLibTypeBase

val _ = new_theory "projectTypes";

val _ = Datatype `platoonLeaderCom = form
   | move
   | secureHalt

val _ = Datatype `omniCom = none
   | omniNA

360
val principal = Datatype `'principal`
  = PlatoonLeader | Omni

val commands = Datatype `'commands`
  = PlatoonLeaderCOM platoonLeaderCom | OmniCOM omniCom

val state = Datatype `'state`
  = MOVE_TO_PB | FORM | MOVE | SECURE_HALT

val output = Datatype `'output`
  = Move_to_ORP | Form | Move | Secure_halt | NoActionTaken | UnAuthenticated | UnAuthorized

(******************************************************************
(* Theorems to prove distinctness and one-to-one. *)
(* *****************************************************************)
val principal_distinct_clauses = distinct_of:`principal`
val principal_distinct_clauses
val platoonLeaderCom_distinct_clauses = distinct_of:`platoonLeaderCom`
val platoonLeaderCom_distinct_clauses
val omniCom_distinct_clauses = distinct_of:`omniCom`
val omniCom_distinct_clauses
val commands_distinct_clauses = distinct_of:`commands`
val commands_distinct_clauses
val state_distinct_clauses = distinct_of:`state`
val state_distinct_clauses
val output_distinct_clauses = distinct_of:`output`
val output_distinct_clauses
val commands_one_one = one_one_of:`commands`
val commands_one_one
val _ = export_theory();
end

(******************************************************************
(* projectUtilitiesScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project’s *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-oneness theorems. *)
(* Project Description: ssmMoveToPB. *)
(******************************************************************)
structure projectUtilitiesScript = struct
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfRules aclrulesTheory aclDrulesTheory satListTheory
open projectTypesTheory
val _ = new_theory "projectUtilities";

(* Functions for extracting commands from input stream. *)
(* Functions for extracting commands from input stream. *)
val getPlatoonLeaderCOM_def = Define ' 
  (getPlatoonLeaderCOM ([]: commands option list) = (NONE: commands option))
  /
  (getPlatoonLeaderCOM ((SOME (PlatoonLeaderCOM cmd))::xs) =
   (SOME (PlatoonLeaderCOM cmd)))
  /
  (getPlatoonLeaderCOM (_::xs) = getPlatoonLeaderCOM xs)'
val getOmniCOM_def = Define ' 
  (getOmniCOM ([]: commands option list) = (NONE: commands option))
  /
  (getOmniCOM ((SOME (OmniCOM cmd))::xs) =
   (SOME (OmniCOM cmd)))
  /
  (getOmniCOM (_::xs) = getOmniCOM xs)'

(* Functions for extracting commands from input list. *)
(* Functions for extracting commands from input list. *)
val getPlatoonLeaderCOMx_def = Define ' 
  (getPlatoonLeaderCOMx ([]: (commands option , principal , 'd , 'e)Form list) = NONE)
  /
  (getPlatoonLeaderCOMx (((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM cmd)))))::xs) =
   (SOME (PlatoonLeaderCOM cmd:commands)))
  /
  (getPlatoonLeaderCOMx (_::xs) = getPlatoonLeaderCOMx xs)'
val getOmniCOMx_def = Define ' 
  (getOmniCOMx ([]: (commands option , principal , 'd , 'e)Form list) = NONE)
  /
  (getOmniCOMx (((Name Omni) says (prop (SOME (OmniCOM cmd)))))::xs) =
   (SOME (OmniCOM cmd:commands)))
  /
  (getOmniCOMx (_::xs) = getOmniCOMx xs)'
val _ = export_theory();
end

D.3.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

(* projectSMScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Description: This file contains the datatype definitions for the project’s *)
(* secure state machine’s next-state and next-output functions. *)
(* Project Description: ssmMoveToPB. *)
(* projectSMScript = struct *)
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssminfRules projectTypesTheory projectUtilitiesTheory
val _ = new_theory "projectSM";

(* Next-state function. *)
val NS_def = Define ' 
  (NS MOVE_TO_PB (exec x) =
if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM form))
then FORM
else MOVE_TO_PB
	
( NS FORM (exec x) =
if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM move))
  then MOVE
  else FORM
	
( NS MOVE (exec x) =
if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM secureHalt))
  then SECURE_HALT
  else MOVE
	
( NS (s:state) (trap _) = s)
( NS (s:state) (discard _) = s) ;

(******************************************************************************)
(* Next-output function. *)
(******************************************************************************)
val NS_def = Define `.
(NOut MOVE_TO_PB (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM form))
    then Form
    else NoActionTaken
	
(NOut FORM (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM move))
    then Move
    else NoActionTaken
	
(NOut MOVE (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM secureHalt))
    then Secure_halt
    else NoActionTaken
	
(NOut (s:state) (trap _) = Unauthorized)
(NOut (s:state) (discard _) = UnAuthenticated) ;

val _ = export_theory();
end

(******************************************************************************)
(* projectSecurityScript.sml *)
(******************************************************************************)
structure projectSecurityScript = struct
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open sminfRules projectTypesTheory projectUtilitiesTheory
val _ = new_theory "projectSecurity";

(******************************************************************************)
(* Authentication definitions. *)
(******************************************************************************)
val authentication_def = Define`
(authentication
  ((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM (x:platoonLeaderCom)))))
  : (commands option ,principal ,'d','e)Form) = T)
(authentication
  ((Name Omni) says (prop (SOME (OmniCOM (x:omniCom)))))
  : (commands option ,principal ,'d','e)Form) = T) ;
(authentication _ = F)

(* stateAuth definitions. *)
(* A state-dependent security context. *)

val stateAuth_def = Define
  stateAuth (s:state) (x:(commands option, principal, 'd,'e)Form list) =
  if (s = MOVE_TO_PB)
    then
      if
        (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM form))
        then
          [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM form))]
          :commands option,principal,'d,'e)Form]
        else
          [prop NONE:commands option,principal,'d,'e)Form]
      else
        if (s = FORM)
          then
            if
              (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM move))
              then
                [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM move))]
                :commands option,principal,'d,'e)Form]
              else
                [prop NONE:commands option,principal,'d,'e)Form]
          else
            if (s = MOVE)
              then
                if
                  (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM secureHalt))
                  then
                    [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM secureHalt))]
                    :commands option,principal,'d,'e)Form]
                  else
                    [prop NONE:commands option,principal,'d,'e)Form]
              else
                [prop NONE:commands option,principal,'d,'e)Form]
            else
              [prop NONE:commands option,principal,'d,'e)Form]"}

(* globalAuth definitions. *)
(* A global security context. *)

val globalAuth_def = Define
  globalAuth (x:(commands option, principal, 'd,'e)Form list) =
  [TT:(commands option, principal, 'd,'e)Form]

val _ = export_theory();

D.3.4.3 ssmMoveToPB Theory: Theorems

(* projectAssuranceExecScript.sml *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the proofs for complete mediation. In particular, these proofs prove that a transition is executed if and only if the input is authenticated and authorized. *)
(* Project Description: ssmMoveToPB.*)

structure projectAssuranceExecScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfrules aclInfrulesTheory aclDrulesTheory satListTheory ssminfTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
open satListTheory bossLib
val _ = new_theory "projectAssuranceExec";
(******************************************************************************)
(* State: MOVE_TO_PB *)
******************************************************************************)

(* Execute transition justified: *)
(* MOVE_TO_PB \rightarrow FORM *)

val helper1 = ISPECL

| :
\`\` authentication : (commands option, principal, \textquoteleft d, e\textquoteright ) Form \rightarrow \textquoteleft bool\textquoteright \),
\`\`globalAuth : (commands option, principal, \textquoteleft d, e\textquoteright ) Form list \rightarrow
(commands option, principal, \textquoteleft d, e\textquoteright ) Form list \textquoteright \,
\`\`stateAuth : state \rightarrow
(commands option, principal, \textquoteleft d, e\textquoteright ) Form list \textquoteright \,
The\textquoteleft\textquoteleft (Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM form)))
\textquoteright\textquoteright :
\`\`ins : (commands option, principal, \textquoteleft d, e\textquoteright ) Form list list \textquoteright \,
\`\`outs : commands option list trType list \textquoteright \]
| TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val MOVE_TO_PB_exec_form_lemma1 =
TAC_PROOF(
([], helper2),
REWRITE_TAC[
CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
satList_CONS, satList_nil,
GSYM satList_conj] - satList - nil,
GSYM satList_conj] - NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
principal_distinct_clauses, commands_distinct_clauses,
omniCom_distinct_clauses, commands_one_one,
state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, state_distinct_clauses,
principal_distinct_clauses, omniCom_distinct_clauses,
satList_CONS, satList_nil] - TR_exec_cmd_rule
THEN REWRITE_TAC[
state_distinct_clauses,
principal_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, principal_distinct_clauses,
principal_distinct_clauses, omniCom_distinct_clauses,
state_distinct_clauses,
principal_distinct_clauses, principal_distinct_clauses]
THEN REWRITE_TAC[
satList_CONS, satList_nil]
THEN PROVE_TAC[
Controls, Modus_Ponens]
)

val _ = save_thm("MOVE_TO_PB_exec_form_lemma1",
MOVE_TO_PB_exec_form_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val MOVE_TO_PB_exec_form_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[
MOVE_TO_PB_exec_form_lemma1,
TR_exec_cmd_rule]
)

val _ = save_thm("MOVE_TO_PB_exec_form_lemma2",
MOVE_TO_PB_exec_form_lemma2)

val MOVE_TO_PB_exec_form_thm =
REWRITE_RULE | propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP | MOVE_TO_PB_exec_form_lemma2
val _ = save_thm("MOVE_TO_PB_exec_form_thm", MOVE_TO_PB_exec_form_thm)

(******************************************************************************
(* State: FORM *)
******************************************************************************
(* Execute transition justified: *)
(* FORM --> MOVE *)
******************************************************************************
val helper1 = ISPEC
  |
  "authentication:{commands option,principal,'d','e}Form --> bool``,
  "globalAuth:{commands option,principal,'d','e}Form list -->
  (commands option,principal,'d','e)Form list``,
  "stateAuth: state -->
  (commands option,principal,'d','e)Form list -->
  (commands option,principal,'d','e)Form list``,
  "((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM move))
  : (commands option,principal,'d','e)Form`)`,
  "ins:{commands option,principal,'d','e}Form list list``,
  "outs:commands option list trType list``,
  TR_exec_cmd_rule
val helper2 = fst(dest_imp(concl helper1))

val FORM_exec_move_lemma1 =
  TAC_PROOF(
    [[]],helper2),
  REWRITE_TAC | CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS, satList Nil,
  GSYM satList conj| THEN
  REWRITE_TAC |
  NOT NONE SOME, NOT SOME NONE, SOME 11, state distinct clauses,
  output distinct clauses, commands distinct clauses,
  principal distinct clauses, platoonLeaderCom distinct clauses,
  omniCom distinct clauses, commands one one,
  GSYM state distinct clauses,
  GSYM output distinct clauses,
  GSYM commands distinct clauses,
  GSYM principal distinct clauses,
  GSYM platoonLeaderCom distinct clauses,
  GSYM omniCom distinct clauses|
  THEN
  REWRITE_TAC |
  satList CONS, satList Nil| THEN
  PROVE_TAC | Controls, Modus_Ponens| )
val _ = save_thm("FORM_exec_move_lemma1", FORM_exec_move_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val FORM_exec_move_lemma2 =
  TAC_PROOF(
    [[]],helper3),
  PROVE_TAC | FORM_exec_move_lemma1,
val FORM_exec_move_thm = REWRITE_RULE [propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]
| FORM_exec_move_lemma2
val _ = save_thm("FORM_exec_move_thm", FORM_exec_move_thm)

(* State: MOVE *)
(* Execute transition justified: *)
(* MOVE --> SECURE_HALT *)
(* *)
val helper1 = ISPECL [
  `(authentication:(commands option , principal , 'd','e)Form -> bool)`,
  `(globalAuth:(commands option , principal , 'd','e)Form list ->
  (commands option , principal , 'd','e)Form list)`,
  `stateAuth: state ->
  (commands option , principal , 'd','e)Form list ->
  (commands option , principal , 'd','e)Form list)`
][
  `(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM secureHalt)))
  :(commands option , principal , 'd','e)Form list list`
][
  `MOVE`,
  `outs:commands option list trType list`]
| TR_exec_cmd_rule
val helper2 = fst(dest_imp(concl helper1))
val MOVE_exec_secureHalt_lemma1 = TAC_PROOF ([[]] , helper2),
rewite_tac [
  CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS , satList_nil ,
  GSYM satList_conj] THEN
rewite_tac [
  NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
  output_distinct_clauses, commands_distinct_clauses,
  principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
  omniCom_distinct_clauses, commands_one_one,
  GSYM state_distinct_clauses,
  GSYM output_distinct_clauses,
  GSYM commands_distinct_clauses,
  GSYM principal_distinct_clauses,
  GSYM platoonLeaderCom_distinct_clauses,
  GSYM omniCom_distinct_clauses] THEN
rewite_tac [satList_CONS , satList_nil] THEN
prove_tac [Controls , Modus_Ponens]
]
val _ = save_thm("MOVE_exec_secureHalt_lemma1",
MOVEXEC_SECUREHALT_LEMMA1)
val helper3 = snd(dest_imp(concl helper1))

val MOVE_exec_secureHalt_lemma2 =
TAC_PROOF(([]].helper3),
PROVE_TAC[MOVE_exec_secureHalt_lemma1,
TR_exec_cmd_rule]

val _ = save_thm("MOVE_exec_secureHalt_lemma2",
MOVE_exec_secureHalt_lemma2)

val MOVE_exec_secureHalt_thm =
REWRITE_RULE[propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
MOVE_exec_secureHalt_lemma2

val _ = save_thm("MOVE_exec_secureHalt_thm",
MOVE_exec_secureHalt_thm)

val _ = export_theory();
end

D.3.5 ssmConductPB

D.3.5.1 ConductPBType Theory: Type Definitions

(*--------------------------------------------------------------------------*)
(* projectTypesScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project's *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-oneness theorems. *)
(* Project Description: ssmConductPB. *)
(*--------------------------------------------------------------------------*)
structure projectTypesScript = struct

open HolKernel Parse boolLib bossLib TypeBase

val _ = new_theory "projectTypes";

val _ = Datatype `platoonLeaderCom
  = secure
  | withdraw
  | complete

val _ = Datatype `platoonSergeantCom
  = actionsIn
  | psgNA

val _ = Datatype `omniCom
  = none
  | omniNA

val _ = Datatype `principal
  = PlatoonLeader
  | PlatoonSergeant
  | Omni

val _ = Datatype `commands
  = PlatoonLeaderCOM platoonLeaderCom

368
val state = Datatype `state
| CONDUCT_PB
| SECURE
| ACTIONS_IN
| WITHDRAW
| COMPLETE

val output = Datatype `output
| Secure
| Withdraw
| Complete
| ActionsIn
| NoActionTaken
| UnAuthenticated
| Unauthorized

{******************************************************************************}
(* Theorems to prove distinctness and one-to-one. *)
******************************************************************************}
val principal_distinct_clauses = distinct_of `principal`
val _ = save_thm("principal_distinct_clauses",
principal_distinct_clauses)

val platoonLeaderCom_distinct_clauses = distinct_of `platoonLeaderCom`
val _ = save_thm("platoonLeaderCom_distinct_clauses",
platoonLeaderCom_distinct_clauses)

val platoonSergeantCom_distinct_clauses = distinct_of `platoonSergeantCom`
val _ = save_thm("platoonSergeantCom_distinct_clauses",
platoonSergeantCom_distinct_clauses)

val omniCom_distinct_clauses = distinct_of `omniCom`
val _ = save_thm("omniCom_distinct_clauses",
omniCom_distinct_clauses)

val commands_distinct_clauses = distinct_of `commands`
val _ = save_thm("commands_distinct_clauses",
commands_distinct_clauses)

val state_distinct_clauses = distinct_of `state`
val _ = save_thm("state_distinct_clauses",
state_distinct_clauses)

val output_distinct_clauses = distinct_of `output`
val _ = save_thm("output_distinct_clauses",
output_distinct_clauses)

val commands_one_one = one_one_of `commands`
val _ = save_thm("commands_one_one",
commands_one_one)

val _ = export_theory();
end

******************************************************************************

structure projectUtilitiesScript = struct
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infrules aclrulesTheory aclDrulesTheory satListTheory
open projectTypesTheory

369
val _ = new_theory "projectUtilities";

(* Functions for extracting commands from input stream. *)
(* *********************************************** *)
val getPlatoonLeaderCOM_def = Define ' 
  (getPlatoonLeaderCOM ([ ]: commands option list) = (NONE: commands option)) /
  (getPlatoonLeaderCOM (( SOME (PlatoonLeaderCOM cmd))::xs) = 
    ( SOME (PlatoonLeaderCOM cmd))) /
  (getPlatoonLeaderCOM (_::xs) = getPlatoonLeaderCOM xs)'

val getPlatoonSergeantCOM_def = Define ' 
  (getPlatoonSergeantCOM ([ ]: commands option list) = (NONE: commands option)) /
  (getPlatoonSergeantCOM (( SOME (PlatoonSergeantCOM cmd))::xs) = 
    ( SOME (PlatoonSergeantCOM cmd))) /
  (getPlatoonSergeantCOM (_::xs) = getPlatoonSergeantCOM xs)'

val getOmniCOM_def = Define ' 
  (getOmniCOM ([ ]: commands option list) = (NONE: commands option)) /
  (getOmniCOM (( SOME (OmniCOM cmd))::xs) = 
    ( SOME (OmniCOM cmd))) /
  (getOmniCOM (_::xs) = getOmniCOM xs)'

(* Functions for extracting commands from input list. *)
(* *********************************************** *)
val getPlatoonLeaderCOMx_def = Define ' 
  (getPlatoonLeaderCOMx ([ ]:(commands option , principal , 'd' e)Form list) = NONE) /
  (getPlatoonLeaderCOMx (((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM cmd))))::xs) = 
    (SOME (PlatoonLeaderCOM cmd:commands))) /
  (getPlatoonLeaderCOMx (_::xs) = getPlatoonLeaderCOMx xs)'

val getPlatoonSergeantCOMx_def = Define ' 
  (getPlatoonSergeantCOMx ([ ]:(commands option , principal , 'd' e)Form list) = NONE) /
  (getPlatoonSergeantCOMx (((Name PlatoonSergeant) says (prop (SOME (PlatoonSergeantCOM cmd))))::xs) = 
    (SOME (PlatoonSergeantCOM cmd:commands))) /
  (getPlatoonSergeantCOMx (_::xs) = getPlatoonSergeantCOMx xs)'

val getOmniCOMx_def = Define ' 
  (getOmniCOMx ([ ]:(commands option , principal , 'd' e)Form list) = NONE) /
  (getOmniCOMx (((Name Omni) says (prop (SOME (OmniCOM cmd))))::xs) = 
    (SOME (OmniCOM cmd:commands))) /
  (getOmniCOMx (_::xs) = getOmniCOMx xs)'

val _ = export_theory();

end

D.3.5.2 ConductPBDDef Theory: Authentication & Authorization Definitions

(* projectSMScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project's *)

370
structure projectSMScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory

val _ = new_theory "projectSM";

(******************************************************************************)
(* Next-state function. *)
(******************************************************************************)
val NS_def = Define'
  (NS CONDUCT_PB (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM secure))
      then SECURE
    else CONDUCT_PB)

  (NS SECURE (exec x) =
    if (getPlatoonSergeantCOM x = SOME (PlatoonSergeantCOM actionsIn))
      then ACTIONS_IN
    else SECURE)

  (NS ACTIONS_IN (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM withdraw))
      then WITHDRAW
    else ACTIONS_IN)

  (NS WITHDRAW (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))
      then COMPLETE
    else WITHDRAW)

  (NS (s : state) (trap _) = s) /
  (NS (s : state) (discard _) = s)"

(******************************************************************************)
(* Next-output function. *)
(******************************************************************************)
val NOut_def = Define'
  (NOut CONDUCT_PB (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM secure))
      then Secure
    else NoActionTaken)

  (NOut SECURE (exec x) =
    if (getPlatoonSergeantCOM x = SOME (PlatoonSergeantCOM actionsIn))
      then ActionsIn
    else NoActionTaken)

  (NOut ACTIONS_IN (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM withdraw))
      then Withdraw
    else NoActionTaken)

  (NOut WITHDRAW (exec x) =
    if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))
      then Complete
    else NoActionTaken)

  (NOut (s : state) (trap _) = Unauthorized) /
  (NOut (s : state) (discard _) = UnAuthenticated)"

val _ = export_theory();
end

(******************************************************************************)

371
structure projectSecurityScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfrules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfrules projectTypesTheory projectUtilitiesTheory

val _ = new theory "projectSecurity";

(* Authentication definitions. *)
(val authentication (((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM (x:platoonLeaderCom)))))
  :(commands option ,principal ,d,e)Form) = T) /
(authentication (((Name PlatoonSergeant) says (prop (SOME (PlatoonSergeantCOM (x:platoonSergeantCom)))))
  :(commands option ,principal ,d,e)Form) = T) /
(authentication (((Name Omni) says (prop (SOME (OmniCOM (x:omniCom)))))
  :(commands option ,principal ,d,e)Form) = T) /
(authentication _ = F)

(* stateAuth definitions. *)
(val stateAuth_def = Define ` stateAuth (s : state ) (x:(commands option ,principal ,d,e)Form list ) =
  if (s = CONDUCT_PB) then
    if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM secure))
      then [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM secure)))
      :(commands option ,principal ,d,e)Form] else [prop NONE:(commands option ,principal ,d,e)Form]
    else if (s = SECURE) then
      if (getPlatoonSergeantCOMx x = SOME (PlatoonSergeantCOM actionsIn))
        then [(Name PlatoonSergeant) controls (prop (SOME (PlatoonSergeantCOM actionsIn)))
        :(commands option ,principal ,d,e)Form] else [prop NONE:(commands option ,principal ,d,e)Form]
    else if (s = ACTIONS_IN) then
      if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM withdraw))
        then [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM withdraw)))
        :(commands option ,principal ,d,e)Form] else [prop NONE:(commands option ,principal ,d,e)Form]
    else if (s = WITHDRAW) then
      if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM complete))
        then [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM complete)))

val globalAuth_def = Define 'globalAuth (x:(commands option , principal , d, e)Form list) =

val _ = export_theory();

end

D.3.5.3 ssmConductPB Theory: Theorems

structure projectAssuranceExecScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssminfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
open satListTheory bossLib

val _ = new_theory "projectAssuranceExec";

val helper1 = ISPECL [
  \''authentication:(commands option , principal , d, e)Form -> bool'\',
  \''globalAuth:(commands option , principal , d, e)Form list ->
  (commands option , principal , d, e)Form list '\',
  "\'stateAuth: state ->
  (commands option , principal , d, e)Format list ->
  (commands option , principal , d, e)Format list '\',
  "\'(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM secure))) :
  (commands option , principal , d, e)Format] '\',
  "\'ins: (commands option , principal , d, e)Format list list '\',
  "\'CONDUCT_PB '
  "\'outs: commands option list trType list ']
  TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val CONDUCT_PB_exec_secure_lemma1 =
  TAC_PROOF( [[]],helper2),
  REWRITE_TAC[
  CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getPlatoonSergeantCOM_def, getOmniCOM_def, getPlatoonLeaderCOMx_def,
  getPlatoonSergeantCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def,
  MAP, propCommandList_def, extractPropCommand_def, satListCONS, satList_nil,
  GSYM satList_conj]
THEN

REWRITE_TAC

NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses, platoonLeaderCom_distinct_clauses, platoonSergeantCom_distinct_clauses, omniCom_distinct_clauses, commands_one_one,

GSYM state_distinct_clauses,

GSYM output_distinct_clauses,

GSYM commands_distinct_clauses,

GSYM principal_distinct_clauses,

GSYM platoonLeaderCom_distinct_clauses,

GSYM platoonSergeantCom_distinct_clauses,

GSYM omniCom_distinct_clauses]

THEN

REWRITE_TAC

satList_CONS, satList_nil]

THEN

PROVE_TAC

[ Controls, Modus_Ponens]

val _ = save_thm("CONDUCT_PB_exec_secure_lemma1", CONDUCT_PB_exec_secure_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val CONDUCT_PB_exec_secure_lemma2 = TAC_PROOF(

([| helper3 |], PROVE_TAC

CONDUCT_PB_exec_secure_lemma1,

TR_exec_cmd_rule)

)

val _ = save_thm("CONDUCT_PB_exec_secure_lemma2", CONDUCT_PB_exec_secure_lemma2)

val CONDUCT_PB_exec_secure_thm = REWRITE_RULE[|

propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]

| CONDUCT_PB_exec_secure_lemma2

val _ = save_thm("CONDUCT_PB_exec_secure_thm", CONDUCT_PB_exec_secure_thm)

(* *******************************************************************************
 * State: SECURE
 * *******************************************************************************
 * Execute transition justified:
 * SECURE ==> ACTIONS_IN
 * *******************************************************)

val helper1 = ISPECL

| `authentication: (commands option, principal, 'd, 'e)Form -> bool`,

`globalAuth: (commands option, principal, 'd, 'e)Form list ->
(commands option, principal, 'd, 'e)Form list```

`stateAuth: state ->
(commands option, principal, 'd, 'e)Form list ->
(commands option, principal, 'd, 'e)Form list```

`[(Name PlatoonSergeant) says (prop (SOME (PlatoonSergeantCOM actionsIn)))]

:(commands option, principal, 'd, 'e)Form```

`ins: (commands option, principal, 'd, 'e)Form list list```

`SECURE```

`outs:commands option list trType list```

| TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))
val SECURE_exec_actionsIn_lemma1 =
TAC_PROOF(
([], helper2),
REWRITE_TAC[
CFGInterp_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getPlatoonSergeantCOM_def, getOmniCOM_def, getPlatoonLeaderCOMx_def,
getPlatoonSergeantCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def,
MAP, propCommandList_def, extractPropCommand_def, extractInput_def,
satList_CONS, satList_nil,
GSYM satList_conj]
THEN
REWRITE_TAC[
NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
platoonSergeantCom_distinct_clauses, omniCom_distinct_clauses,
commands_one_one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM platoonSergeantCom_distinct_clauses,
GSYM omniCom_distinct_clauses]
THEN
REWRITE_TAC[
satList_CONS, satList_nil]
THEN
PROVE_TAC[
Controls, Modus_Ponens]
)

val _ = save_thm("SECURE_exec_actionsIn_lemma1",
SECURE_exec_actionsIn_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val SECURE_exec_actionsIn_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[
SECURE_exec_actionsIn_lemma1,
TR_exec_cmd_rule]
)

val _ = save_thm("SECURE_exec_actionsIn_lemma2",
SECURE_exec_actionsIn_lemma2)

val SECURE_exec_actionsIn_thm =
REWRITE_RULE[
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
SECURE_exec_actionsIn_lemma2

val _ = save_thm("SECURE_exec_actionsIn_thm",
SECURE_exec_actionsIn_thm)

(* ***************************************************************************)
(* State: ACTIONS_IN *)
(* ***************************************************************************)
(* Execute transition justified: * ACTIONS_IN --> WITHDRAW *)
(* ***************************************************************************)
val helper1 = ISPECL
| `authentication:(commands option, principal,'d', 'e)Form -> bool`,
`globalAuth:(commands option, principal,'d', 'e)Form list ->
(commands option, principal,'d', 'e)Form list`,
`stateAuth: state ->

375
(commands option, principal, 'd', 'e) Form list ->
(commands option, principal, 'd', 'e) Form list

``(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM withdraw)))
: (commands option, principal, 'd', 'e) Form list`
``
``ins: (commands option, principal, 'd', 'e) Form list list``
``ACTIONs_IN``
``outs: commands option list trType list``
| TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val ACTIONS_IN_exec_withdraw_lemma1 =
TAC_PROOF(
([], helper2),
REWRITE_TAC[CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getPlatoonSergeantCOM_def, getOmniCOM_def, getPlatoonLeaderCOMx_def,
getPlatoonSergeantCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def,
MAP, propCommandList_def, extractPropCommand_def, satList CONS, satList nil,
GSYM satList conj]
THEN
REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME 11, state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
platoonSergeantCom_distinct_clauses, omniCom_distinct_clauses,
commands one one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM platoonSergeantCom_distinct_clauses,
GSYM omniCom_distinct_clauses]
THEN
REWRITE_TAC[satList CONS, satList nil]
THEN
PROVE_TAC[Controls, Modus_Ponens]
)

val _ = save_thm("ACTIONS_IN_exec_withdraw_lemma1",
ACTIONS_IN_exec_withdraw_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val ACTIONS_IN_exec_withdraw_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[ACTIONS_IN_exec_withdraw_lemma1,
TR_exec_cmd_rule]
)

val _ = save_thm("ACTIONS_IN_exec_withdraw_lemma2",
ACTIONS_IN_exec_withdraw_lemma2)

val ACTIONS_IN_exec_withdraw_thm =
REWRITE_RULE[propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
| ACTIONS_IN_exec_withdraw_lemma2

val _ = save_thm("ACTIONS_IN_exec_withdraw_thm",
ACTIONS_IN_exec_withdraw_thm)

(*****************************************)
(* State: WITHDRAW *)
(*****************************************)
val helper1 = ISPECL
```
authentication : (commands option , principal , 'd' , 'e') Form --> bool
```
\[\text{(commands option , principal , 'd' , 'e') Form --> boole}
\]
\[\text{(commands option , principal , 'd' , 'e') Form list --> boole}
\]
\[\text{(commands option , principal , 'd' , 'e') Form list --> boole}
\]
\[\text{((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM complete)))
\]
\[\text{:((commands option , principal , 'd' , 'e') Form)]
```
\[\text{ins : (commands option , principal , 'd' , 'e') Form list list
\]
\[\text{WITHDRAW
\]
\[\text{outs : commands option list trType list
```
| TR_exec_cmd_rule
```
val helper2 = fst (dest_imp (concl helper1))
```
val WITHDRAW_exec_complete_lemma1 = TAC_PROOF(
[[]] , helper2).
```
\[\text{CFGInterpret_def , globalAuth_def , stateAuth_def , getPlatoonLeaderCOM_def ,
\[\text{getPlatoonSergeantCOM_def , getOmniCOM_def , getPlatoonLeaderCOMx_def ,
\[\text{getPlatoonSergeantCOMx_def , getOmniCOMx_def , inputList_def , extractInput_def ,
\[\text{MAP, propCommandList_def , extractPropCommand_def , satList_CONS , satList_nil ,
\[\text{GSYM satList_conj}\]
```
| TR_rewrite Tac
```
\[\text{NOT_NONE_SOME , NOT_SOME_NONE , SOME_11 , state_distinct_clauses ,
\[\text{output_distinct_clauses , commands_distinct_clauses ,
\[\text{principal_distinct_clauses , platoonLeaderCom_distinct_clauses ,
\[\text{platoonSergeantCom_distinct_clauses , omniCom_distinct_clauses ,
\[\text{commands_one_one ,
\[\text{GSYM state_distinct_clauses ,
\[\text{GSYM output_distinct_clauses ,
\[\text{GSYM commands_distinct_clauses ,
\[\text{GSYM principal_distinct_clauses ,
\[\text{GSYM platoonLeaderCOM_distinct_clauses ,
\[\text{GSYM platoonSergeantCOM_distinct_clauses ,
\[\text{GSYM omniCOM_distinct_clauses}\]
```
| TR_rewrite Tac
```
\[\text{satList_CONS , satList_nil}\]
```
| TR_rewrite Tac
```
\[\text{Controls , Modus_Ponens}\]
```
|)
```
val __ = save_thm ("WITHDRAW_exec_complete_lemma1", WITHDRAW_exec_complete_lemma1)
```
val helper3 = snd (dest_imp (concl helper1))
```
val WITHDRAW_exec_complete_lemma2 = TAC_PROOF(
[[]] , helper3).
```
\[\text{WITHDRAW_exec_complete_lemma1 ,
```
\[\text{TR_exec_cmd_rule}\]
```
|)
```
val __ = save_thm ("WITHDRAW_exec_complete_lemma2", WITHDRAW_exec_complete_lemma2)
```
val WITHDRAW_exec_complete_thm = REWRITE_RULE[
```
\[\text{propCommandList_def , inputList_def , extractPropCommand_def , extractInput_def ,
```
|)
D.4 Vertical Slice

D.4.1 ssmSecureHalt

D.4.1.1 SecureHalt_Type Theory: Type Definitions

(* ***********************************************************************)
(* projectTypesScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project’s *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-oneness theorems. *)
(* Project Description: ssmSecureHalt. *)
(* ***********************************************************************)
structure projectTypesScript = struct
  open HolKernel Parse boolLib bossLib TypeBase
  val _ = new_theory "projectTypes";
  val _ = Datatype `platoonLeaderCom
    = secure
    | orpRecon
    | withdraw
    | complete
  `;
  val _ = Datatype `omniCom
    = none
    | omniNA
  `;
  val _ = Datatype `principal
    = PlatoonLeader
    | Omni
  `;
  val _ = Datatype `commands
    = PlatoonLeaderCOM platoonLeaderCom
    | OmniCOM omniCom
  `;
  val _ = Datatype `state
    = SECURE_HALTS
    | SECURE
    | ORP_RECON
    | WITHDRAW
    | COMPLETE
  `;
  val _ = Datatype `output
    = Secure_halts
    | Secure
    | OrpRecon
    | Withdraw
    | Complete
    | NoActionTaken
    | UnAuthenticated

val _ = save_thm("WITHDRAW_exec_complete_thm",
               WITHDRAW_exec_complete_thm)
val _ = export_theory();
end
structure projectUtilitiesScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfRules aclrulesTheory aclDrulesTheory satListTheory
open projectTypesTheory

val _ = new_theory "projectUtilities";

(* Functions for extracting commands from input stream. *)

val getPlatoonLeaderCOM_def = Define ' 
  (getPlatoonLeaderCOM ([]:commands option list) = (NONE:commands option)) 
  /\ (getPlatoonLeaderCOM ((SOME (PlatoonLeaderCOM cmd))::xs) = 
    (SOME (PlatoonLeaderCOM cmd))) 
  /\ (getPlatoonLeaderCOM (_::xs) = getPlatoonLeaderCOM xs)'

val getOmniCOM_def = Define ' 
  (getOmniCOM ([]:commands option list) = (NONE:commands option)) 
  /\ (getOmniCOM ((SOME (OmniCOM cmd))::xs) = 
    (SOME (OmniCOM cmd))) 
  /\ (getOmniCOM (_::xs) = getOmniCOM xs)'

end

 val principal_distinct_clauses = distinct_of `principal`
 val _ = save_thm("principal_distinct_clauses", principal_distinct_clauses)

 val platoonLeaderCom_distinct_clauses = distinct_of `platoonLeaderCom`
 val _ = save_thm("platoonLeaderCom_distinct_clauses", platoonLeaderCom_distinct_clauses)

 val omniCom_distinct_clauses = distinct_of `omniCom`
 val _ = save_thm("omniCom_distinct_clauses", omniCom_distinct_clauses)

 val commands_distinct_clauses = distinct_of `commands`
 val _ = save_thm("commands_distinct_clauses", commands_distinct_clauses)

 val state_distinct_clauses = distinct_of `state`
 val _ = save_thm("state_distinct_clauses", state_distinct_clauses)

 val output_distinct_clauses = distinct_of `output`
 val _ = save_thm("output_distinct_clauses", output_distinct_clauses)

 val commands_one_one = one_one_of `commands`
 val _ = save_thm("commands_one_one", commands_one_one)
Functions for extracting commands from input list.

val getPlatoonLeaderCOMx_def = Define ` (getPlatoonLeaderCOMx ([[]:commands option, principal, 'd', 'e)Form list) = NONE) /
 (getPlatoonLeaderCOMx (((Name PlatoonLeader) says (prop (SOME {PlatoonLeaderCOM cmd})))::xs) = (SOME {PlatoonLeaderCOM cmd:commands})) /
 (getPlatoonLeaderCOMx (_::xs) = getPlatoonLeaderCOMx xs)`

val getOmniCOMx_def = Define ` (getOmniCOMx ([[]:commands option, principal, 'd', 'e)Form list) = NONE) /
 (getOmniCOMx (((Name Omni) says (prop (SOME {OmniCOM cmd})))::xs) = (SOME {OmniCOM cmd:commands})) /
 (getOmniCOMx (_::xs) = getOmniCOMx xs)`

val _ = export_theory();
end

D.4.1.2 SecureHaltDef Theory: Authentication & Authorization Definitions

structure projectSMScript = struct
open HolKernel Parse boolLib bossLib TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory
val _ = new_theory "projectSM";

val NS_def = Define ` (NS SECURE_HALT (exec x) =
 if (getPlatoonLeaderCOM x = SOME {PlatoonLeaderCOM secure})
 then SECURE
 else SECURE_HALT
) /
 (NS SECURE (exec x) =
 if (getPlatoonLeaderCOM x = SOME {PlatoonLeaderCOM orpRecon})
 then ORP_RECON
 else SECURE
) /
 (NS ORP_RECON (exec x) =
 if (getPlatoonLeaderCOM x = SOME {PlatoonLeaderCOM withdraw})
 then WITHDRAW
 else ORP_RECON
) /
 (NS WITHDRAW (exec x) =
 if (getPlatoonLeaderCOM x = SOME {PlatoonLeaderCOM complete})
 then COMPLETE
 else WITHDRAW
) /
 (NS (s:state) (trap _) = s) /
 (NS (s:state) (discard _) = s)`
Next–output function.

val NS_def = Define
(NOut SECURE_HALT (exec x) = 
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM secure))
  then Secure
  else NoActionTaken
) /
(NOut SECURE (exec x) = 
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM orpRecon))
  then OrpRecon
  else NoActionTaken
) /
(NOut ORP_RECON (exec x) = 
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM withdraw))
  then Withdraw
  else NoActionTaken
) /
(NOut WITHDRAW (exec x) = 
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))
  then Complete
  else NoActionTaken
) /
(NOut (s : state) (trap _) = UnAuthorized) /
(NOut (s : state) (discard _) = UnAuthenticated)

val _ = export_theory();
end

(val authentication_def = Define
((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM (x:platoonLeaderCom)))))
 : (commands option, principal, 'd', 'e') Form) = T) /
(authentication ((Name Omni) says (prop (SOME (OmniCOM (x:omniCom)))))
 : (commands option, principal, 'd', 'e') Form) = T) /
(authentication _ = F)

(val stateAuth_def = Define
stateAuth (s : state) (x: (commands option, principal, 'd', 'e') Form list) = 
  if (s = SECURE_HALT)
  then

Authentication definitions.

stateAuth definitions.

A state–dependent security context.
(getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM secure))
then
[(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM secure)))
 : (commands option, principal, 'd', 'e')Form]
else
[prop NONE: (commands option, principal, 'd', 'e')Form]
else
  if (s = SECURE)
  then
    (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM orpRecon))
    then
      [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM orpRecon)))
       : (commands option, principal, 'd', 'e')Form]
    else
      [prop NONE: (commands option, principal, 'd', 'e')Form]
  else
    if (s = ORP_RECON)
    then
      (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM withdraw))
      then
        [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM withdraw)))
         : (commands option, principal, 'd', 'e')Form]
      else
        [prop NONE: (commands option, principal, 'd', 'e')Form]
    else
      if (s = WITHDRAW)
      then
        (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM complete))
        then
          [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM complete)))
           : (commands option, principal, 'd', 'e')Form]
        else
          [prop NONE: (commands option, principal, 'd', 'e')Form]
      else
        if (s = WITHDRAW)
        then
          (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM complete))
          then
            [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM complete)))
             : (commands option, principal, 'd', 'e')Form]
          else
            [prop NONE: (commands option, principal, 'd', 'e')Form]
        else
          [prop NONE: (commands option, principal, 'd', 'e')Form]

****

val globalAuth_def = Define 'globalAuth (x:(commands option, principal, 'd', 'e')Form list) =
[TT:(commands option, principal, 'd', 'e')Form]

val _ = export_theory ();
end

D.4.1.3  ssmsSecureHalt Theory: Theorems

**************
(* projectAssuranceExecScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the proofs for complete mediation. In particular, these proofs prove that a transition is executed if and only if the input is authenticated and authorized. *)
(* Project Description: ssmsSecureHalt *)
**************
structure projectAssuranceExecScript = struct
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmsTheory
open ssminfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
open satListTheory bossLib
val _ = new_theory "projectAssuranceExec";
Execute transition justified:
SECURE_HALT --> SECURE

```ml
val helper1 = ISPECL
  [
    `authentication : (commands option, principal, 'd', 'e)Form -> bool`,
    `globalAuth : (commands option, principal, 'd', 'e)Form list ->
      (commands option, principal, 'd', 'e)Form list`,
    `stateAuth : state ->
      (commands option, principal, 'd', 'e)Form list ->
      (commands option, principal, 'd', 'e)Form list`,
    `[(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM secure)))] :
      (commands option, principal, 'd', 'e)Form list list`,
    `SECURE_HALT`,
    `outs : commands option list `,
    `ins : (commands option, principal, 'd', 'e)Form list list`,
  ]
  TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val SECURE_HALT_exec_secure_lemma1 =
  TAC_PROOF( [[]], helper2),
  REWRITE_TAC[CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
      getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
      extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
      satList_CONS, satList_nil, GSYM satList_conj] THEN
  REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
      output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses,
      platoonLeaderCOM_distinct_clauses, omniCom_distinct_clauses, commands_one_one,
      GSYM state_distinct_clauses, GSYM output_distinct_clauses,
      GSYM commands_distinct_clauses, GSYM principal_distinct_clauses,
      GSYM platoonLeaderCOM_distinct_clauses, GSYM omniCom_distinct_clauses] THEN
  REWRITE_TAC[satList_CONS, satList_nil] THEN
  PROVE_TAC[Controls, Modus_Ponens]
)

val _ = save_thm("SECURE_HALT_exec_secure_lemma1",
SECURE_HALT_exec_secure_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val SECURE_HALT_exec_secure_lemma2 =
  TAC_PROOF( [[]], helper3),
  PROVE_TAC[SECURE_HALT_exec_secure_lemma1,
SECURE_HALT_exec_secure_lemma1, TR_exec_cmd_rule]
)

val _ = save_thm("SECURE_HALT_exec_secure_lemma2",
SECURE_HALT_exec_secure_lemma2)

val SECURE_HALT_exec_secure_thm =
  REWRITE_RULE[

```

383
```
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP
| SECURE_HALT_exec_secure lemma2

val _ = save_thm("SECURE_HALT_exec_secure_thm",
SECURE_HALT_exec_secure_thm)

(* State: SECURE *)

(* Execute transition justified: SECURE --> ORP_RECON *)

val helper1 = ISPECL
[ "authentication : (commands option , principal , d , e) Form -> bool",
  globalAuth : (commands option , principal , d , e) Form list ->
  (commands option , principal , d , e) Form list t",
  stateAuth : state ->
  (commands option , principal , d , e) Form list ->
  (commands option , principal , d , e) Form list t",
  "[(Name PlatoonLeader) says (prop SOME (PlatoonLeaderCOM orpRecon))]
  :(commands option , principal , d , e) Form]",
  "ins : (commands option , principal , d , e) Form list list",
  "outs : commands option list trType list"
] TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val SECURITY_exec_orpRecon_lemma1 =
TAC_PROOF(
([],helper2),
REWRITE_TAC [CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
satList_CONS, satList_nil,
GSYM satList_conj]
THEN
REWRITE_TAC
NOT_NONE_SOME, NOT_SOME_NONE, SOME_1, state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
omniCom_distinct_clauses, commands_one_one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM omniCom_distinct_clauses]
THEN
REWRITE_TAC[satList_CONS, satList_nil]
THEN
PROVE_TAC[Controls, Modus_Ponens]
)

val _ = save_thm("SECURE_exec_orpRecon_lemma1",
SECURE_exec_orpRecon_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val SECURITY_exec_orpRecon_lemma2 =
TAC_PROOF(
([],helper3),
PROVE_TAC[SECURE_exec_orpRecon_lemma1, TR_exec_cmd_rule]
)
val _ = save_thm("SECURE_exec_orpRecon_lemma2", SECURE_exec_orpRecon_lemma2)

val SECURE_exec_orpRecon_thm = REWRITE_RULE[
  propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP
]SECURE_exec_orpRecon_lemma2
val _ = save_thm("SECURE_exec_orpRecon_thm", SECURE_exec_orpRecon_thm)

("***************
(* State: ORP_RECON

(* Execute transition justified: *)
(* ORP_RECON --> WITHDRAW *)

val helper1 = ISPECL[
  "authentication:(commands option, principal , 'd', 'e')Form -> bool",
  "globalAuth:(commands option, principal , 'd', 'e')Form list ->
  (commands option, principal , 'd', 'e')Form list ",
  "stateAuth: state ->
  (commands option, principal , 'd', 'e')Form list ->
  (commands option, principal , 'd', 'e')Form list ",
  "[)(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM withdraw)))
  :(commands option, principal , 'd', 'e)Form]",
  "ins:(commands option, principal , 'd', 'e)Form list list ",
  "ORP_RECON ",
  "outs:commands option list trType list "]

| TR_exec_cmd_rule
val helper2 = fst(dest_imp(concl helper1))

val ORP_RERCON_exec_withdraw_lemma1 =
  TAC_PROOF(
    [], helper2)

REWRITE_TAC[
  CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS, satList_nil,
  GSYM satList_conj]

THEN
REWRITE_TAC[
  NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
  output_distinct_clauses, commands_distinct_clauses,
  principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
  omniCom_distinct_clauses, commands_one_one,
  GSYM state_distinct_clauses,
  GSYM output_distinct_clauses,
  GSYM commands_distinct_clauses,
  GSYM principal_distinct_clauses,
  GSYM platoonLeaderCom_distinct_clauses,
  GSYM omniCom_distinct_clauses]

THEN
REWRITE_TAC[
  satList_CONS, satList_nil]

THEN
PROVE_TAC[
  Controls , Modus_Ponens] )

val _ = save_thm("ORP_RECON_exec_withdraw_lemma1", ORP_RECON_exec_withdraw_lemma1)

385
val helper3 = snd(dest_imp(concl helper1))

val ORP_RECON_exec_withdraw_lemma2 = TAC_PROOF(
  ([], helper3),
  PROVE_TAC |
  ORP_RECON_exec_withdraw_lemma1,
  TR_exec_cmd_rule |
)

val _ = save_thm("ORP_RECON_exec_withdraw_lemma2",
  ORP_RECON_exec_withdraw_lemma2)

val ORP_RECON_exec_withdraw_thm = REWRITE_RULE |
  propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
  MAP |
  ORP_RECON_exec_withdraw_lemma2

val _ = save_thm("ORP_RECON_exec_withdraw_thm",
  ORP_RECON_exec_withdraw_thm)

(*******************************************************************************)
(* State : WITHDRAW *)
(*******************************************************************************)
(* Execute transition justified: *)
(* WITHDRAW --> COMPLETE *)
(*******************************************************************************)

val helper1 = ISPECL |
  [| 'authentication:(commands option , principal , 'd, 'e)Form -> bool',
     'globalAuth:(commands option , principal , 'd, 'e)Form list ->
       (commands option , principal , 'd, 'e)Form list',
     'stateAuth: state ->
       (commands option , principal , 'd, 'e)Form list ->
       (commands option , principal , 'd, 'e)Form list',
     '((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM complete)))
       : (commands option , principal , 'd, 'e)Form)',
     'ins:(commands option , principal , 'd, 'e)Form list list',
     'outs:commands option list trType list'] |
  TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val WITHDRAW_exec_complete_lemma1 = TAC_PROOF(
  ([], helper2),
  REWRITE_TAC |
  CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS, satList_nil,
  GSYM satList_conj |
  THEN
  REWRITE_TAC |
  NOT_SOME_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
  output_distinct_clauses, commands_distinct_clauses,
  principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
  omniCom_distinct_clauses, commands_one_one,
  GSYM state_distinct_clauses,
  GSYM output_distinct_clauses,
  GSYM commands_distinct_clauses,
  GSYM principal_distinct_clauses,
  GSYM platoonLeaderCom_distinct_clauses,
  GSYM omniCom_distinct_clauses |
  THEN
  REWRITE_TAC |
  satList_CONS, satList_nil |
  THEN

386
val helper3 = snd(dest_imp(concl helper1))

val WITHDRAW_exec_complete_lemma2 = TAC_PROOF([[]; helper3], PROVE_TAC [WITHDRAW_exec_complete_lemma1, TR_exec_cmd_rule])

val _ = save_thm("WITHDRAW_exec_complete_lemma2", WITHDRAW_exec_complete_lemma2)

val WITHDRAW_exec_complete_thm = REWRITE_RULE [propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP] WITHDRAW_exec_complete_lemma2

val _ = save_thm("WITHDRAW_exec_complete_thm", WITHDRAW_exec_complete_thm)

val _ = export_theory();

end

D.4.2 ssmORPRecon

D.4.2.1 ORPReconType Theory: Type Definitions

(* ******************************* *)
(* projectTypesScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project's *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-oneness theorems. *)
(* Project Description: ssmORPRecon. *)
(* ******************************* *)
structure projectTypesScript = struct

open HolKernel Parse boolLib bossLib TypeBase

val _ = new_theory "projectTypes";

val _ = Datatype `platoonLeaderCom
  = contingencyPlan
  | mowToORP
  | conductORP
  | formST
  | returnToUnit
  | complete

val _ = Datatype `omniCom
  = none
  | omniNA

val _ = Datatype `principal
structure projectUtilitiesScript = struct

val = Datatype `commands
| PlatoonLeaderCOM platoonLeaderCom
| OmniCOM omniCom

val = Datatype `state
= ORP RECON
| CONTINGENCY PLAN
| MOVE TO ORP
| CONDUCT ORP
| FORM ST
| RETURN TO UNIT
| COMPLETE

val = Datatype `output
= ContingencyPlan
| MoveToORP
| ConductORP
| FormST
| ReturnToUnit
| Complete
| NoActionTaken
| UnAuthenticated
| Unauthorized

(* Theorems to prove distinctness and one-to-one. *)
(* ********************************************************************* *)
val principal_distinct_clauses = distinct_of `:principal`
val = save_thm("principal_distinct_clauses",
principal_distinct_clauses)

val platoonLeaderCom_distinct_clauses = distinct_of `:platoonLeaderCom`
val = save_thm("platoonLeaderCom_distinct_clauses",
platoonLeaderCom_distinct_clauses)

val omniCom_distinct_clauses = distinct_of `:omniCom`
val = save_thm("omniCom_distinct_clauses",
omniCom_distinct_clauses)

val commands_distinct_clauses = distinct_of `:commands`
val = save_thm("commands_distinct_clauses",
commands_distinct_clauses)

val state_distinct_clauses = distinct_of `:state`
val = save_thm("state_distinct_clauses",
state_distinct_clauses)

val output_distinct_clauses = distinct_of `:output`
val = save_thm("output_distinct_clauses",
output_distinct_clauses)

val commands_one_one = one_one_of `:commands`
val = save_thm("commands_one_one",
commands_one_one)

val = export_theory();
end

(* ********************************************************************* *)
(* projectUtilitiesScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the datatype definitions for the project’s *)
(* secure state machine: principals, commands, states, and outputs. It also *)
(* contains the distinctness and one-to-one theorems. *)
(* Project Description: ssmORPRecon *)
(* ********************************************************************* *)
D.4.2.2 ORPReconDef Theory: Authentication & Authorization Definitions
(* Next-state function. *)
val NS_def = Define'

(NS ORP_RECON (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM contingencyPlan))
    then CONTINGENCY_PLAN
  else ORP_RECON)
\/

(NS CONTINGENCY_PLAN (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM moveToORP))
    then MOVE_TO_ORP
  else CONTINGENCY_PLAN)
\/

(NS MOVE_TO_ORP (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM conductORP))
    then CONDUCT_ORP
  else MOVE_TO_ORP)
\/

(NS CONDUCT_ORP (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM formST))
    then FORM_ST
  else CONDUCT_ORP)
\/

(NS FORM_ST (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM returnToUnit))
    then RETURN_TO_UNIT
  else FORM_ST)
\/

(NS RETURN_TO_UNIT (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))
    then COMPLETE
  else RETURN_TO_UNIT)
\/

(NS (s : state) (trap _) = s) /\
:NS (s : state) (discard _) = s'

(* Next-output function. *)
val NS_def = Define'

(NOut ORP_RECON (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM contingencyPlan))
    then ContingencyPlan
  else NoActionTaken)
\/

(NOut CONTINGENCY_PLAN (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM moveToORP))
    then MoveToORP
  else NoActionTaken)
\/

(NOut MOVE_TO_ORP (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM conductORP))
    then ConductORP
  else NoActionTaken)
\/

(NOut CONDUCT_ORP (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM formST))
    then FormST
  else NoActionTaken)
\/

(NOut FORM_ST (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM returnToUnit))
    then ReturnToUnit
  else NoActionTaken)
\/

(NOut RETURN_TO_UNIT (exec x) =
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))
    then Complete
  else NoActionTaken)
\/

(NOut (s : state) (trap _) = Unauthorized) /\
(NOut (s : state) (discard _) = UnAuthenticated)

val _ = export_theory();
end

(******************************************************************************)
(* projectSecurityScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Stokke *)
(* Description: This file contains the security policy (state-dependent *)
(* authorization, global authorization) and authentication policy. *)
(* Project Description: ssmORPRecon. *)
(******************************************************************************)
structure projectSecurityScript = struct

open HOLKernel Parse boolLib bossLib TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmRules projectTypesTheory projectUtilitiesTheory

val _ = new_theory "projectSecurity";

(******************************************************************************)
(* Authentication definitions. *)
(******************************************************************************)
val authentication_def = Define `'authentication``
  ((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM (x:platoonLeaderCom))))
  :(commands option, principal, 'd, 'e)Form) = T) /
  (authentication``
  ((Name Omni) says (prop (SOME (OmniCOM (x:omniCom))))
  :(commands option, principal, 'd, 'e)Form) = T) /
  (authentication_ = F)``

(******************************************************************************)
(* stateAuth definitions. *)
(******************************************************************************)
val stateAuth_def = Define `stateAuth``
  (s : state) (x:(commands option, principal, 'd, 'e)Form list) =
  if (s = ORP_RECON)
  then
      if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM contingencyPlan))
      then
      [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM contingencyPlan))))
      :(commands option, principal, 'd, 'e)Form]
      else
      [prop NONE:(commands option, principal, 'd, 'e)Form]
  else
  if (s = CONTINGENCY_PLAN)
  then
  if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM moveToORP))
  then
  [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM moveToORP))))
  :(commands option, principal, 'd, 'e)Form]
  else
  [prop NONE:(commands option, principal, 'd, 'e)Form]
  else
  if (s = MOVE_TO_ORP)
  then
  if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM conductORP))
  then
  [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM conductORP))))
  :(commands option, principal, 'd, 'e)Form]
  else
  [prop NONE:(commands option, principal, 'd, 'e)Form]
  else
  if (s = CONDUCT_ORP)

391
then
    if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM formST))
        then
            [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM formST)))
            : (commands option, principal, 'd', 'e)Form]
        else
            [prop NONE: (commands option, principal, 'd', 'e)Form]
    else
        if (s = FORM_ST)
            then
                if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM returnToUnit))
                    then
                        [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM returnToUnit)))
                        : (commands option, principal, 'd', 'e)Form]
                    else
                        [prop NONE: (commands option, principal, 'd', 'e)Form]
                else
                    if (s = RETURN_TO_UNIT)
                        then
                            if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM complete))
                                then
                                    [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM complete)))
                                    : (commands option, principal, 'd', 'e)Form]
                                else
                                    [prop NONE: (commands option, principal, 'd', 'e)Form]
                        else
                            [prop NONE: (commands option, principal, 'd', 'e)Form]
        else
            [prop NONE: (commands option, principal, 'd', 'e)Form]

val globalAuth_def = Define 

projectAssuranceExecScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
open satListTheory bossLib

val = new_theory "projectAssuranceExec";

val helper1 = ISPECL
val helper2 = fst(dest_imp(concl helper1))

val ORP_RECON_exec_contingencyPlan_lemma1 =
TAC_PROOF(
([], helper2),
REWRITE_TAC[CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def, getOmnIOM_def, getPlatoonLeaderCOMx_def, getOmnIOMx_def, inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS, satList_nil, GSYM satList_conj]
THEN REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses, platoonLeaderCom_distinct_clauses, OmniCom_distinct_clauses, commands_one_one, GSYM state_distinct_clauses, GSYM output_distinct_clauses, GSYM commands_distinct_clauses, GSYM principal_distinct_clauses, GSYM platoonLeaderCom_distinct_clauses, GSYM OmniCom_distinct_clauses]
THEN REWRITE_TAC[satList_CONS, satList_nil]
THEN
PROVE_TAC[Controls, Modus_Ponens]
)

val _ = save_thm("ORP_RECON_exec_contingencyPlan_lemma1", ORP_RECON_exec_contingencyPlan_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val ORP_RECON_exec_contingencyPlan_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[ORP_RECON_exec_contingencyPlan_lemma1, TR_exec_cmd_rule]
)

val _ = save_thm("ORP_RECON_exec_contingencyPlan_lemma2", ORP_RECON_exec_contingencyPlan_lemma2)

val ORP_RECON_exec_contingencyPlan_thm =
REWRITE_RULE[propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]
ORP_RECON_exec_contingencyPlan_lemma2

val _ = save_thm("ORP_RECON_exec_contingencyPlan_thm", ORP_RECON_exec_contingencyPlan_thm)
val helper1 = ISPECL
[ `(authentication : (commands option , principal , 'd', 'e') Form -> bool)` ,
  `(globalAuth : (commands option , principal , 'd', 'e') Form list ->
  (commands option , principal , 'd', 'e') Form list)` ,
  `(stateAuth : state ->
  (commands option , principal , 'd', 'e') Form list ->
  (commands option , principal , 'd', 'e') Form list)` ,
  `((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM moveToORP))))
  : (commands option , principal , 'd', 'e') Form list list)` ,
  `(CONTINGENCY_PLAN ,
  `ins : (commands option , principal , 'd', 'e') Form list list ,
  `outs : commands option list trType list` `] TR_exec_cmd_rule
val helper2 = fst (dest_imp (concl helper1 ))

val CONTINGENCY_PLAN_exec_moveToORP_lemma1 =
TAC_PROOF(
  [[[helper2] ],
   REWRITE_TAC
   CFGInterpret_def ,
   globalAuth_def ,
   stateAuth_def ,
   getPlatoonLeaderCOM_def ,
   getOmniCOM_def ,
   getPlatoonLeaderCOMx_def ,
   getOmniCOMx_def ,
   inputList_def ,
   extractInput_def ,
   MAP ,
   propCommandList_def ,
   extractPropCommand_def ,
   satList_CONS ,
   satList_nil ,
   GSYM satList_conj]
  THEN
   REWRITE_TAC
   NOT_NONE_SOME , NOT_SOME_NONE , SOME_11 ,
   state_distinct_clauses ,
   output_distinct_clauses ,
   commands_distinct_clauses ,
   principal_distinct_clauses ,
   platoonLeaderCom_distinct_clauses ,
   omniCom_distinct_clauses ,
   commands_one_one ,
   GSYM state_distinct_clauses ,
   GSYM output_distinct_clauses ,
   GSYM commands_distinct_clauses ,
   GSYM principal_distinct_clauses ,
   GSYM platoonLeaderCom_distinct_clauses ,
   GSYM omniCom_distinct_clauses]
  THEN
   REWRITE_TAC
   satList_CONS ,
   satList_nil]
  THEN
   PROVE_TAC
   [ Controls ,
   Modus_Ponens ]
)

val _ = save_thm ("CONTINGENCY_PLAN_exec_moveToORP_lemma1" ,
  CONTINGENCY_PLAN_exec_moveToORP_lemma1)

val helper3 = snd (dest_imp (concl helper1 ))

val CONTINGENCY_PLAN_exec_moveToORP_lemma2 =
TAC_PROOF(
  [[[helper3] ],
   PROVE_TAC
   CONTINGENCY_PLAN_exec_moveToORP_lemma1 ,
   TR_exec_cmd_rule]
)

val _ = save_thm ("CONTINGENCY_PLAN_exec_moveToORP_lemma2" ,
  CONTINGENCY_PLAN_exec_moveToORP_lemma2)

val CONTINGENCY_PLAN_exec_moveToORP_thm =
REWRITE_RULE]

394
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP
| CONTINGENCY_PLAN_exec_moveToORP_lemma2

val _ = save_thm("CONTINGENCY_PLAN_exec_moveToORP_thm",
CONTINGENCY_PLAN_exec_moveToORP_thm)

(* State: MOVE_TO_ORP *)

(* Execute transition justified: *)

val helper1 = ISPECL

``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
``
```

val helper2 = fst(dest_imp(concl helper1))

val MOVE_TO_ORP_exec_conductORP_lemma1 =
TAC_PROOF(
([], helper2),
REWRITE_TAC[
CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
satList_CONS, satList_nil,
GSYM satList_conj]
THEN
REWRITE_TAC[
NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
omniCom_distinct_clauses, commands_one_one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM omniCom_distinct_clauses]
THEN
REWRITE_TAC[
satList_CONS, satList_nil]
THEN
PROVE_TAC[
Controls, Modus_Ponens]
)

val _ = save_thm("MOVE_TO_ORP_exec_conductORP_lemma1",
MOVE_TO_ORP_exec_conductORP_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val MOVE_TO_ORP_exec_conductORP_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[
MOVE_TO_ORP_exec_conductORP_lemma1,
TR_exec_cmd_rule]"
val _ = save_thm("MOVE_TO_ORP_exec_conductORP_lemma2", 
        MOVE_TO_ORP_exec_conductORP_lemma2)

val MOVE_TO_ORP_exec_conductORP_thm = 
    REWRITE_RULE[
        propCommandList_def , inputList_def , extractPropCommand_def , extractInput_def , MAP |
        MOVE_TO_ORP_exec_conductORP_lemma2
    ]

val _ = save_thm("MOVE_TO_ORP_exec_conductORP_thm", 
        MOVE_TO_ORP_exec_conductORP_thm)

(* ****************************************************************************** *)
(* State: CONDUCT_ORP                                                        *)
(* ****************************************************************************** *)
(* Execute transition justified: *)
(* CONDUCT_ORP -> FORM_ST *)
(* ****************************************************************************** *)
val helper1 = ISPECL
    [authentication : (commands option , principal , 'd','e)Form -> bool',
    'globalAuth : (commands option , principal , 'd','e)Form list ->
    (commands option , principal , 'd','e)Form list',
    'stateAuth : state ->
    (commands option , principal , 'd','e)Form list ->
    (commands option , principal , 'd','e)Form list',
    'ins : (commands option , principal , 'd','e)Form list list',
    'outs : commands option list trType list',
    'CONDUCT_ORP',
    ''Connor : commands option list trType list']
| TR_exec_cmd_rule
val helper2 = fst(dest_imp(concl helper1))

val CONDUCT_ORP_exec_formST_lemma1 = 
    TAC_PROOF(
        ([],helper2),
        REWRITE_TAC[
            CFGInterpret_def , globalAuth_def , stateAuth_def , getPlatoonLeaderCOM_def ,
            getOmniCOM_def , getPlatoonLeaderCOMx_def , getOmniCOMx_def , inputList_def ,
            extractInput_def , MAP , propCommandList_def , extractPropCommand_def ,
            satList_CONS , satList_nil ,
            GSYM state_distinct_clauses ,
            GSYM output_distinct_clauses ,
            GSYM principal_distinct_clauses ,
            GSYM platoonLeaderCom_distinct_clauses ,
            GSYM omniCom_distinct_clauses] THEN
            REWRITE_TAC[
                NOT_NONE SOME , NOT SOME_NONE , SOME_11 , state_distinct_clauses ,
                output_distinct_clauses , commands_distinct_clauses ,
                principal_distinct_clauses , platoonLeaderCom_distinct_clauses ,
                omniCom_distinct_clauses , commands_one_one ,
                GSYM state_distinct_clauses ,
                GSYM output_distinct_clauses ,
                GSYM commands_distinct_clauses ,
                GSYM principal_distinct_clauses ,
                GSYM platoonLeaderCom_distinct_clauses ,
                GSYM omniCom_distinct_clauses] THEN
            REWRITE_TAC[
                satList_CONS , satList_nil] THEN
            PROVE_TAC[
                Controls , Modus_Ponens]
        )

val _ = save_thm("CONDUCT_ORP_exec_formST_lemma1", 
        CONDUCT_ORP_exec_formST_lemma1)
val helper3 = snd(dest_imp(concl helper1))

val CONDUCT_ORP_exec_formST_lemma2 = TAC_PROOF(
  ([]; helper3),
  PROVE_TAC[
  CONDUCT_ORP_exec_formST_lemma1,
  TR_exec_cmd_rule]
)

val _ = save_thm("CONDUCT_ORP_exec_formST_lemma2",
  CONDUCT_ORP_exec_formST_lemma2)

val CONDUCT_ORP_exec_formST_thm = REWRITE_RULE[
  propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
  MAP
]CONDUCT_ORP_exec_formST_lemma2

val _ = save_thm("CONDUCT_ORP_exec_formST_thm",
  CONDUCT_ORP_exec_formST_thm)

(*---------------------------------------------------------------*)
(* State : FORM_ST *)
(*---------------------------------------------------------------*)
(* Execute transition justified: *)
(* FORM_ST ==> RETURN_TO_UNIT *)
(*---------------------------------------------------------------*)

val helper1 = ISPECL[
  (commands option , principal , 'd', 'e')Form -> bool,
  (commands option , principal , 'd', 'e')Form list ->
  (commands option , principal , 'd', 'e')Form list ->
  (commands option , principal , 'd', 'e')Form list ->
  (commands option , principal , 'd', 'e')Form list ->
  (commands option , principal , 'd', 'e')Form list ->
  (Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM returnToUnit)))
  : (commands option , principal , 'd', 'e')Form]

val helper2 = fst(dest_imp(concl helper1))

val FORM_ST_exec_returnToUnit_lemma1 = TAC_PROOF(
  ([]; helper2),
  REWRITE_TAC[
  CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
  getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS, satList_nil,
  GSYM satList_conj
]THEN
  REWRITE_TAC[
  NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
  output_distinct_clauses, commands_distinct_clauses,
  principal_distinct_clauses, platoonLeaderCOM_distinct_clauses,
  omniCom_distinct_clauses, commands_one_one,
  GSYM state_distinct_clauses,
  GSYM output_distinct_clauses,
  GSYM commands_distinct_clauses,
  GSYM principal_distinct_clauses,
  GSYM platoonLeaderCOM_distinct_clauses,
  GSYM omniCom_distinct_clauses]
]THEN
  REWRITE_TAC[
  satList_CONS, satList_nil]
]THEN

397
val helper3 = snd(dest_imp(concl helper1))

val FORM_ST_exec_returnToUnit_lemma2 = TAC_PROOF(
  ([], helper3),
  PROVE_TAC [FORM_ST_exec_returnToUnit_lemma1, TR_exec_cmd_rule]
)

val = save_thm("FORM_ST_exec_returnToUnit_lemma2",
  FORM_ST_exec_returnToUnit_lemma2)

val FORM_ST_exec_returnToUnit_thm = REWRITE_RULE [propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]

| FORM_ST_exec_returnToUnit_lemma1


val helper1 = ISPECL

| `
` authentication:(commands option,principal,'d','e)Form -> bool`,
` globalAuth:(commands option,principal,'d','e)Form list ->
(commands option,principal,'d','e)Form list`;
` stateAuth: state ->
(commands option,principal,'d','e)Form list ->
(commands option,principal,'d','e)Form list`;
` ((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM complete))))
 : (commands option,principal,'d','e)Form)`;
` ins: (commands option,principal,'d','e)Form list list`;
` outs: commands option list trType list`;
| TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val RETURN_TO_UNIT_exec_complete_lemma1 = TAC_PROOF(
  ([], helper2),
  REWRITE_TAC [CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def, getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS, satList_nil, GSYM satList_conj]
)

THEN

REWRITE_TAC

NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses, platoonLeaderCom_distinct_clauses, omniCom_distinct_clauses, commands_one_one, GSYM state_distinct_clauses, GSYM output_distinct_clauses,
D.4.3 ssmMoveToORP4L

D.4.3.1 MoveToORP4L Type Theory: Type Definitions

```plaintext
structure projectTypesScript = struct
```

```plaintext
open HolKernel Parse boolLib bossLib TypeBase
```

```plaintext
val _ = new_theory "projectTypes";
```

```plaintext
val _ = Datatype `platoonLeaderCom
```

---

GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM omniCom_distinct_clauses;

THEN

REWRITE_TAC satList_CONS, satList nil;

THEN

PROVE_TAC [Controls, Modus_Ponens];

val _ = save_thm ("RETURN_TO_UNIT_exec_complete_lemma1",
RETURN_TO_UNIT_exec_complete_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val RETURN_TO_UNIT_exec_complete_lemma2 =

TAC_PROOF (RETURN_TO_UNIT_exec_complete_lemma1,
TR_exec_cmd_rule)

val _ = save_thm ("RETURN_TO_UNIT_exec_complete_lemma2",
RETURN_TO_UNIT_exec_complete_lemma2)

val RETURN_TO_UNIT_exec_complete_thm =

REWRITE_RULE [propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]

RETURN_TO_UNIT_exec_complete_thm2

val _ = save_thm ("RETURN_TO_UNIT_exec_complete_thm",
RETURN_TO_UNIT_exec_complete_thm)

val _ = export_theory();

end
```
val _ = Datatype `omniCom
  = none
  | omniNA
val _ = Datatype `principal
  = PlatoonLeader
  | Omni
val _ = Datatype `commands
  = PlatoonLeaderCOM platoonLeaderCom
  | OmniCOM omniCom
val _ = Datatype `state
  = MOVE_TO_ORP
  | FORM_RT
  | RT_MOVE
  | RT_HALT
  | COMPLETE
val _ = Datatype `output
  = FormRT
  | RtMove
  | RtHalt
  | Complete
  | NoActionTaken
  | Unauthorized

(* Theorems to prove distinctness and one-to-one. *)
(* Theorems to prove distinctness and one-to-one. *)
val principal_distinct_clauses = distinct_of `principal`
val _ = save_thm("principal_distinct_clauses",
            principal_distinct_clauses)
val platoonLeaderCom_distinct_clauses = distinct_of `platoonLeaderCom`
val _ = save_thm("platoonLeaderCom_distinct_clauses",
              platoonLeaderCom_distinct_clauses)
val omniCom_distinct_clauses = distinct_of `omniCom`
val _ = save_thm("omniCom_distinct_clauses",
                omniCom_distinct_clauses)
val commands_distinct_clauses = distinct_of `commands`
val _ = save_thm("commands_distinct_clauses",
                commands_distinct_clauses)
val state_distinct_clauses = distinct_of `state`
val _ = save_thm("state_distinct_clauses",
                state_distinct_clauses)
val output_distinct_clauses = distinct_of `output`
val _ = save_thm("output_distinct_clauses",
                output_distinct_clauses)
val commands_one_one = one_one_of `commands`
val _ = save_thm("commands_one_one",
                commands_one_one)
val _ = export_theory();
end
structure projectUtilitiesScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory
open projectTypesTheory

val _ = new_theory "projectUtilities";

(striped functions for extracting commands from input stream.

val getPlatoonLeaderCOM_def = Define ' 
  (getPlatoonLeaderCOM ([]:commands option list) = (NONE:commands option))
  (getPlatoonLeaderCOM ((SOME (PlatoonLeaderCOM cmd))::xs) = 
    (SOME (PlatoonLeaderCOM cmd)))
  (getPlatoonLeaderCOM (__::xs) = getPlatoonLeaderCOM xs)' 

val getOmniCOM_def = Define ' 
  (getOmniCOM ([]:commands option list) = (NONE:commands option))
  (getOmniCOM ((SOME (OmniCOM cmd))::xs) = 
    (SOME (OmniCOM cmd)))
  (getOmniCOM (__::xs) = getOmniCOM xs)' 

(striped functions for extracting commands from input list.

val getPlatoonLeaderCOMx_def = Define ' 
  (getPlatoonLeaderCOMx ([]:(commands option,principal,d,e)Form list) = NONE)
  (getPlatoonLeaderCOMx (((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM cmd)))))::xs) = 
    (SOME (PlatoonLeaderCOM cmd:commands)))
  (getPlatoonLeaderCOMx (__::xs) = getPlatoonLeaderCOMx xs)' 

val getOmniCOMx_def = Define ' 
  (getOmniCOMx ([]:(commands option,principal,d,e)Form list) = NONE)
  (getOmniCOMx (((Name Omni) says (prop (SOME (OmniCOM cmd)))))::xs) = 
    (SOME (OmniCOM cmd:commands)))
  (getOmniCOMx (__::xs) = getOmniCOMx xs)' 

val _ = export_theory();

end

D.4.3.2 MoveToORP4LDef Theory: Authentication & Authorization
Definitions

(striped projectSMScript.sml)

structure projectSMScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory

val _ = new_theory "projectSM";
(** Next-state function. **)  
(``val NS_def = Define` 
(`NS MOVE_TO_ORP (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM formRT))  
  then FORM_RT  
  else MOVE_TO_ORP` 
)` 
(`NS FORM_RT (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM rtMove))  
  then RT_MOVE  
  else FORM_RT` 
)` 
(`NS RT_MOVE (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM rtHalt))  
  then RT_HALT  
  else RT_MOVE` 
)` 
(`NS RT_HALT (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))  
  then COMPLETE  
  else RT_HALT` 
)` 
(`NS (s : state) (trap _) = s` 
)` 
(`NS (s : state) (discard _) = s` 
)`

(** Next-output function. **)  
(``val NS_def = Define` 
(``NOut MOVE_TO_ORP (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM formRT))  
  then FormRT  
  else NoActionTaken` 
)` 
(``NOut FORM_RT (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM rtMove))  
  then RtMove  
  else NoActionTaken` 
)` 
(``NOut RT_MOVE (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM rtHalt))  
  then RtHalt  
  else NoActionTaken` 
)` 
(``NOut RT_HALT (exec x) =  
  if (getPlatoonLeaderCOM x = SOME (PlatoonLeaderCOM complete))  
  then Complete  
  else NoActionTaken` 
)` 
(``NOut (s : state) (trap _) = UnAuthorized` 
)` 
(``NOut (s : state) (discard _) = UnAuthenticated` 
)`

``val _ = export_theory();``
end
``

(** projectSecurityScript.sml **)  
(``val _ = export_theory();``)

(** Description: This file contains the security policy (state-dependent authorization, global authorization) and authentication policy. **)  
(``structure projectSecurityScript = struct``)
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open aclInfRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open sssInfRules projectTypesTheory projectUtilitiesTheory

val _ = new theory "projectSecurity";

(* Authentication definitions. *)

val authentication_def = Define ' (authentication
  ((((Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM (x:platoonLeaderCom)))))
  :(commands option ,principal ,d,e)Form) = T) /
  (authentication
  ((((Name Omni) says (prop (SOME (OmniCOM (x:omniCom)))))
  :(commands option ,principal ,d,e)Form) = T) /
  (authentication_ = F))'

(* stateAuth definitions. *)

val stateAuth_def = Define ' stateAuth (s:state) (x:(commands option ,principal ,d,e)Form list) =
  if (s = MOVE_TO_ORP)
  then
    if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM formRT))
    then
      [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM formRT)))
      :(commands option ,principal ,d,e)Form]
    else
      [prop NONE:(commands option ,principal ,d,e)Form]
  else
    if (s = FORM_RT)
    then
      if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM rtMove))
      then
        [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM rtMove)))
        :(commands option ,principal ,d,e)Form]
      else
        [prop NONE:(commands option ,principal ,d,e)Form]
    else
      if (s = RT_MOVE)
      then
        if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM rtHalt))
        then
          [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM rtHalt)))
          :(commands option ,principal ,d,e)Form]
        else
          [prop NONE:(commands option ,principal ,d,e)Form]
      else
        if (s = RT_HALT)
        then
          if (getPlatoonLeaderCOMx x = SOME (PlatoonLeaderCOM complete))
          then
            [(Name PlatoonLeader) controls (prop (SOME (PlatoonLeaderCOM complete)))
            :(commands option ,principal ,d,e)Form]
          else
            [prop NONE:(commands option ,principal ,d,e)Form]
        else
          [prop NONE:(commands option ,principal ,d,e)Form]'

(* globalAuth definitions. *)

val globalAuth_def = Define ' globalAuth (x:(commands option ,principal ,d,e)Form list) =
  [TT:(commands option ,principal ,d,e)Form]
D.4.3.3 ssmMoveToORP4L Theory: Theorems

(structure projectAssuranceExecScript =
  struct
  open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
  open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
  open ssminfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
  open satListTheory bossLib
  val _ = new_theory "projectAssuranceExec";
)

(* State: MOVE_TO_ORP *)

(* Execute transition justified: *)

val helper1 = ISPECL |
  `authentications: (commands option , principal , 'd', 'e) Form -> bool` ,
  `globalAuth: (commands option , principal , 'd', 'e) Form list ->
  (commands option , principal , 'd', 'e) Form list` ,
  `stateAuth: state ->
  (commands option , principal , 'd', 'e) Form list ->
  (commands option , principal , 'd', 'e) Form list` ,
  `[(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM formRT)))
  : (commands option , principal , 'd', 'e) Form]` ,
  `ins: (commands option , principal , 'd', 'e) Form list list` ,
  `outs: commands option list trType list` |
  TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val MOVE_TO_ORP_exec_formRT_lemma1 =
  TAC_PROOF([[] , helper2] ,
    REWRITE_TAC CFGInterpret_def , globalAuth_def , stateAuth_def , getPlatoonLeaderCOM_def ,
    getOmniCOM_def , getPlatoonLeaderCOMx_def , getOmniCOMx_def , inputList_def ,
    extractInput_def , MAP , propCommandList_def , extractPropCommand_def ,
    satList_CONS , satList_nil ,
    GSYM satList_conj]
  THEN
    REWRITE_TAC
    NOT_NONE SOME NOT SOME NONE SOME_11 state_distinct_clauses ,
    output_distinct_clauses , commands_distinct_clauses ,
    principal_distinct_clauses , platoonLeaderCom_distinct_clauses ,
    OmniCom_distinct_clauses , commands_one_one ,
    GSYM state_distinct_clauses ,
    GSYM output_distinct_clauses ,
    GSYM principal_distinct_clauses ,
    GSYM platoonLeaderCom_distinct_clauses ,
    GSYM OmniCom_distinct_clauses]
  THEN

val _ = export_theory();
end
REWRITE_TAC[
satList_CONS, satList_nil]
THEN
PROVE_TAC[
Controls, Modus_Ponens]
)

val _ = save_thm("MOVE_TO_ORP_exec_formRT_lemma1",
MOVE_TO_ORP_exec_formRT_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val MOVE_TO_ORP_exec_formRT_lemma2 =
TAC_PROOF(
[[]; helper3],
PROVE_TAC[
MOVE_TO_ORP_exec_formRT_lemma1,
TR_exec_cmd_rule]
)

val _ = save_thm("MOVE_TO_ORP_exec_formRT_lemma2",
MOVE_TO_ORP_exec_formRT_lemma2)

val MOVE_TO_ORP_exec_formRT_thm =
REWRITE_RULE[
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP
] MOVE_TO_ORP_exec_formRT_lemma2
val _ = save_thm("MOVE_TO_ORP_exec_formRT_thm",
MOVE_TO_ORP_exec_formRT_thm)

(* State: FORM_RT *)

(* Execute transition justified: *)

val helper1 = ISPECL
[`
 authentication:(commands option, principal,'d','e)'Form -> bool`,
`globalAuth:(commands option, principal,'d','e)'Form list ->
(commands option, principal,'d','e)'Form list`,
`stateAuth: state ->
(commands option, principal,'d','e)'Form list ->
(commands option, principal,'d','e)'Form list`,
`([Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM rtMove))): (commands option, principal,'d','e)'Form`,
`ins:(commands option, principal,'d','e)'Form list list`,
`FORM_RT`,
`outs:commands option list trType list`]
TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val FORM_RT_exec_rtMove_lemma1 =
TAC_PROOF(
[[]; helper2],
REWRITE_TAC[
CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def,
getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def,
extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS, satList_nil,
GSYM satList_conj]
THEN
REWRITE_TAC[
NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
omniCom_distinct_clauses, commands_one_one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM omniCom_distinct_clauses
THEN
REWRITE_TAC[
satList_CONS, satList_nil]
THEN
PROVE_TAC[
Controls, Modus_Ponens]
)

val _ = save_thm("FORM_RT_exec_rtMove_lemma1",
FORM_RT_exec_rtMove_lemma1)

val helper3 = snd(dest_imp(concl helper1 ))

val FORM_RT_exec_rtMove_lemma2 =
TAC_PROOF(
([],helper3),
PROVE_TAC[
FORM_RT_exec_rtMove_lemma1,
TR_exec_cmd_rule]
)

val _ = save_thm("FORM_RT_exec_rtMove_lemma2",
FORM_RT_exec_rtMove_lemma2)

val FORM_RT_exec_rtMove_thm =
REWRITE_RULE[
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
FORM_RT_exec_rtMove_lemma2
val _ = save_thm("FORM_RT_exec_rtMove_thm",
FORM_RT_exec_rtMove_thm)

(* State: RT_MOVE  *)
(* Execute transition justified: *)
(* RT_MOVE --> RT_HALT *)

val helper1 = ISPECL[

```
"authentication:(commands option,principal,'d','e)Form -> bool",
"globalAuth:(commands option,principal,'d','e)Form list ->
(commands option,principal,'d','e)Form list`",
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
extractInput_def, propCommandList_def, extractPropCommand_def,
satList_CONS, satList_nil,
GSYM satList_conj
THEN
REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses,
output_distinct_clauses, commands_distinct_clauses,
principal_distinct_clauses, platoonLeaderCom_distinct_clauses,
omniCom_distinct_clauses, commands_one_one,
GSYM state_distinct_clauses,
GSYM output_distinct_clauses,
GSYM commands_distinct_clauses,
GSYM principal_distinct_clauses,
GSYM platoonLeaderCom_distinct_clauses,
GSYM omniCom_distinct_clauses]
THEN
REWRITE_TAC[satList_CONS, satList_nil]
THEN
PROVE_TAC[Controls, Modus_Ponens])
val = save_thm("RT_MOVE_exec_rtHalt_lemma1",
RT_MOVE_exec_rtHalt_lemma1)

val helper3 = snd(dest_imp(concl helper1))
val RT_MOVE_exec_rtHalt_lemma2 =
TAC_PROOF(
([], helper3),
PROVE_TAC[RT_MOVE_exec_rtHalt_lemma1,
TRExec_cmd_rule])
val = save_thm("RT_MOVE_exec_rtHalt_lemma2",
RT_MOVE_exec_rtHalt_lemma2)

val RT_MOVE_exec_rtHalt_thm =
REWRITE_RULE[propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
| RT_MOVE_exec_rtHalt_lemma2
val = save_thm("RT_MOVE_exec_rtHalt_thm",
RT_MOVE_exec_rtHalt_thm)

(* State: RT_HALT *)
(* Execute transition justified: RT_HALT --> COMPLETE *)
val helper1 = ISPECL
| 'authentication:(commands option, principal, 'd,'e)Form -> bool',
| 'globalAuth:(commands option, principal, 'd,'e)Form list ->
| (commands option, principal, 'd,'e)Form list ',
| 'stateAuth: state ->
| (commands option, principal, 'd,'e)Form list ->
| (commands option, principal, 'd,'e)Form list ',
| [(Name PlatoonLeader) says (prop (SOME (PlatoonLeaderCOM complete))]
| (commands option, principal, 'd,'e)Form'],
| 'ins:(commands option, principal, 'd,'e)Form list list ',
| 'RT_HALT ',
| 'outs:commands option list trType list '
| TRExec_cmd_rule
val helper2 = fst(dest_imp(concl helper1))
val RT_HALT_exec_complete_lemma1 = TAC_PROOF(([] : helper2), REWRITE_TAC [CFGInterpret_def, globalAuth_def, stateAuth_def, getPlatoonLeaderCOM_def, getOmniCOM_def, getPlatoonLeaderCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS, satList_nil, GSYM satList_conj]
THEN REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses, platoonLeaderCom_distinct_clauses, omniCom_distinct_clauses, commands_one_one, GSYM state_distinct_clauses, GSYM output_distinct_clauses, GSYM commands_distinct_clauses, GSYM principal_distinct_clauses, GSYM platoonLeaderCom_distinct_clauses, GSYM omniCom_distinct_clauses]
THEN REWRITE_TAC[satList_CONS, satList_nil]
THEN PROVE_TAC[Controls, Modus_Ponens]
)
val _ = save_thm("RT_HALT_exec_complete_lemma1", RT_HALT_exec_complete_lemma1)

val helper3 = snd(dest_imp(concl helper1))
val RT_HALT_exec_complete_lemma2 = TAC_PROOF(([] : helper3), PROVE_TAC[RT_HALT_exec_complete_lemma1, TR_exec_cmd_rule])
val _ = save_thm("RT_HALT_exec_complete_lemma2", RT_HALT_exec_complete_lemma2)

val RT_HALT_exec_complete_thm = REWRITE_RULE[propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]
| RT_HALT_exec_complete_lemma2
val _ = save_thm("RT_HALT_exec_complete_thm", RT_HALT_exec_complete_thm)
val _ = export_theory();
end

D.4.4 ssmFormRT

D.4.4.1 FormRTType Theory: Type Definitions

(* ******************************************************* *)
(* * projectTypesScript.sml * *)

408
structure projectTypesScript = struct

open HolKernel Parse boolLib bossLib TypeBase

val _ = new_theory "projectTypes";

val _ = Datatype `squadLeaderCom
  = rtPosition
  | rtOrient
  | rtAlert
  | complete

val _ = Datatype `omniCom
  = none
  | omniNA

val _ = Datatype `principal
  = SquadLeader
  | Omni

val _ = Datatype `commands
  = SquadLeaderCOM squadLeaderCom
  | OmniCOM omniCom

val _ = Datatype `state
  = RT_FORM
  | RT_POSITION
  | RT_ORIENT
  | RT_ALERT
  | COMPLETE

val _ = Datatype `output
  = RtPosition
  | RtOrient
  | RtAlert
  | Complete
  | NoActionTaken
  | UnAuthenticated
  | Unauthorized

val principal_distinct_clauses = distinct_of`principal`
val _ = save_thm("principal_distinct_clauses",
  principal_distinct_clauses)

val squadLeaderCom_distinct_clauses = distinct_of`squadLeaderCom`
val _ = save_thm("squadLeaderCom_distinct_clauses",
  squadLeaderCom_distinct_clauses)

val omniCom_distinct_clauses = distinct_of`omniCom`
val _ = save_thm("omniCom_distinct_clauses",
  omniCom_distinct_clauses)

val commands_distinct_clauses = distinct_of`commands`
val _ = save_thm("commands_distinct_clauses",
  commands_distinct_clauses)

val state_distinct_clauses = distinct_of`state`
val _ = save_thm("state_distinct_clauses",
  state_distinct_clauses)

val output_distinct_clauses = distinct_of`output`
val _ = save_thm("output_distinct_clauses", 

409
structure projectUtilitiesScript = struct

open HolKernel Parse boolLib boolLib bossLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory
open projectTypesTheory

val = new_theory "projectUtilities";

(******************************************************************************)
(* Functions for extracting commands from input stream. *)
(******************************************************************************)
val getSquadLeaderCOM_def = Define ' 
  (getSquadLeaderCOM ([]:commands option list) = (NONE:commands option))
  /
  (getSquadLeaderCOM ((SOME (SquadLeaderCOM cmd)):: xs) =
    (SOME (SquadLeaderCOM cmd)))
  /
  (getSquadLeaderCOM (_:: xs) = getSquadLeaderCOM xs)
val getOmniCOM_def = Define ' 
  (getOmniCOM ([]:commands option list) = (NONE:commands option))
  /
  (getOmniCOM ((SOME (OmniCOM cmd)):: xs) =
    (SOME (OmniCOM cmd)))
  /
  (getOmniCOM (_:: xs) = getOmniCOM xs)

(******************************************************************************)
(* Functions for extracting commands from input list. *)
(******************************************************************************)
val getSquadLeaderCOMx_def = Define ' 
  (getSquadLeaderCOMx ([]:(commands option , principal , 'd', 'e)Form list) = NONE)
  /
  (getSquadLeaderCOMx (((Name SquadLeader) says (prop (SOME (SquadLeaderCOM cmd))))):: xs) =
    (SOME (SquadLeaderCOM cmd:commands)))
  /
  (getSquadLeaderCOMx (_:: xs) = getSquadLeaderCOMx xs)
val getOmniCOMx_def = Define ' 
  (getOmniCOMx ([]:(commands option , principal , 'd', 'e)Form list) = NONE)
  /
  (getOmniCOMx (((Name Omni) says (prop (SOME (OmniCOM cmd)))):: xs) =
    (SOME (OmniCOM cmd:commands)))
  /
  (getOmniCOMx (_:: xs) = getOmniCOMx xs)

val = export_theory();
end
D.4.4.2 FormRTDef Theory: Authentication & Authorization Definitions

(******************************************************************************)
(* projectSMScript.sml *)
(* Date: 1 August 2018 *)
(* Author: Assured Things by Design: Lori Pickering, Keara Hill, and Keaten *)
(* Description: This file contains the datatype definitions for the project's *)
(* secure state machine's next-state and next-output functions. *)
(* Project Description: ssmFormRT. *)
(******************************************************************************)

structure projectSMScript = struct

open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssminfRules projectTypesTheory projectUtilitiesTheory

val _ = new_theory "projectSM";

(******************************************************************************)
(* Next-state function. *)
(******************************************************************************)

val NS_def = Define`
  (NS RT FORM (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtPosition))
   then RT_POSITION
   else RT_FORM)
  /
  (NS RT POSITION (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtOrient))
   then RT_ORIENTATION
   else RT_POSITION)
  /
  (NS RT ORIENT (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtAlert))
   then RT_ALERT
   else RT_ORIENTATION)
  /
  (NS RT ALERT (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM complete))
   then COMPLETE
   else RT_ALERT)
  /
  (NS (s:state) (trap _) = s) /
  (NS (s:state) (discard _) = s)`

(******************************************************************************)
(* Next-output function. *)
(******************************************************************************)

val NOut NS_def = Define`
  (NOut RT FORM (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtPosition))
   then RT Position
   else NoActionTaken)
  /
  (NOut RT POSITION (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtOrient))
   then RT ORIENT
   else NoActionTaken)
  /
  (NOut RT ORIENT (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM rtAlert))
   then RT ALERT
   else NoActionTaken)
  /
  (NOut RT ALERT (exec x) =
   if (getSquadLeaderCOM x = SOME (SquadLeaderCOM complete))
   then Complete
   else NoActionTaken)
  /
  (NOut (s:state) (trap _) = UnAuthorized) /

411
```
val _ = export_theory();
end

structure projectSecurityScript = struct

open HolKernel Parse boolLib bossLib TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open sminfRules projectTypesTheory projectUtilitiesTheory
val _ = new_theory "projectSecurity";

(******************************************************************************)
(* Authentication definitions. *)
(******************************************************************************)
val authentication_def = Define ` authentication ((Name SquadLeader) says (prop (SOME (SquadLeaderCOM x:squadLeaderCom)))) : (commands option,principal,'d,'e)Form = T) /
(authentication ((Name Omni) says (prop (SOME (OmniCOM x:omniCom)))) : (commands option,principal,'d,'e)Form = T) /
(authentication _ = F)```

(******************************************************************************)
(* stateAuth definitions. *)
(******************************************************************************)
val stateAuth_def = Define ` stateAuth (s : state) (x :(commands option,principal,'d,'e)Form list) = if (s = RT_FORM)
then
  if (getSquadLeaderCOMx x = SOME (SquadLeaderCOM rtPosition))
  then
    [(Name SquadLeader) controls (prop (SOME (SquadLeaderCOM rtPosition)))) : (commands option,principal,'d,'e)Form]
  else
    [prop NONE :(commands option,principal,'d,'e)Form]
else
  if (s = RT_POSITION)
then
  if (getSquadLeaderCOMx x = SOME (SquadLeaderCOM rtOrient))
  then
    [(Name SquadLeader) controls (prop (SOME (SquadLeaderCOM rtOrient)))) : (commands option,principal,'d,'e)Form]
  else
    [prop NONE :(commands option,principal,'d,'e)Form]
else
  if (s = RT_ORIENTATION)
then
  if (getSquadLeaderCOMx x = SOME (SquadLeaderCOM rtAlert))
  then
    [(Name SquadLeader) controls (prop (SOME (SquadLeaderCOM rtAlert)))) : (commands option,principal,'d,'e)Form]
  else
    [prop NONE :(commands option,principal,'d,'e)Form]
else
  if (s = RT_ALERT)
```

(OUT (s : state) (discard _) = UnAuthenticated)'

412
then
  if
    (getSquadLeaderCOM x = SOME (SquadLeaderCOM complete))
  then
    [(Name SquadLeader) controls (prop (SOME (SquadLeaderCOM complete)))
      : (commands option, principal, 'd', 'e') Form]
  else
    [prop NONE : (commands option, principal, 'd', 'e') Form]
else
  [prop NONE : (commands option, principal, 'd', 'e') Form]


D.4.4.3 ssmFormRT Theory: Theorems

_OVERRIDE

structure projectAssuranceExecScript = struct
open HolKernel Parse boolLib bossLib TypeBase TypeBase listTheory optionTheory
open acl_infRules aclrulesTheory aclDrulesTheory satListTheory ssmTheory
open ssmInfRules projectTypesTheory projectUtilitiesTheory projectSecurityTheory
open satListTheory bossLib
val _ = new_theory "projectAssuranceExec";

(* State: RT_FORM *)
(* Execute transition justified: *)
(* RT_FORM --> RT_POSITION *)
val helper1 = ISPECL
  ["authentication : (commands option, principal, 'd', 'e') Form \rightarrow bool",
   "globalAuth : (commands option, principal, 'd', 'e') Form list \rightarrow
   (commands option, principal, 'd', 'e') Form list ",
   "stateAuth : state \rightarrow
   (commands option, principal, 'd', 'e') Form list \rightarrow
   (commands option, principal, 'd', 'e') Form list ",
   "[(Name SquadLeader) says (prop (SOME (SquadLeaderCOM rtPosition)))
     : (commands option, principal, 'd', 'e') Form]",
   "ins : (commands option, principal, 'd', 'e') Form list list ",
   "RT_FORM",
   "outs : commands option list trType list"]
  TR_exec_cmd_rule
val helper2 = fst (dest_imp (concl helper1))
val RT_FORM_exec_rtPosition_lemma1 =
  TAC_PROOF
    ([], helper2),
  REWRITE_TAC [413]
val helper1 = ISPECL
[[(Name SquadLeader) says (prop (SOME (SquadLeaderCOM rtOrient)))
: (commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
, stateAuth: state ->
(commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
,'[(Name SquadLeader) says (prop (SOME (SquadLeaderCOM rtOrient)))
: (commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
,'ins:(commands option , principal , 'd', 'e)Form list list ->
'RT_POSITION'].
'outs:commands option list trType list`

(val helper3 = snd(dest_imp(concl helper1)))

(val RT_FORM_exec_rtPosition_lemma2 =
TAC_PROOF(
([], helper3).
PROVE_TAC
RT_FORM_exec_rtPosition_lemma1,
TR_exec_cmd_rule)
)

(val = save_thm("RT_FORM_exec_rtPosition_lemma2",
RT_FORM_exec_rtPosition_lemma2)

(val RT_FORM_exec_rtPosition_thm =
REWRITE_RULE[
propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
MAP]
| RT_FORM_exec_rtPosition_lemma2
val = save_thm("RT_FORM_exec_rtPosition_thm",
RT_FORM_exec_rtPosition_thm)

(***********************************************************************)
(* State: RT_POSITION *)
(***********************************************************************)

(val helper1 = ISPECL
[[(Name SquadLeader) says (prop (SOME (SquadLeaderCOM rtOrient)))
: (commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
, stateAuth: state ->
(commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
,'[(Name SquadLeader) says (prop (SOME (SquadLeaderCOM rtOrient)))
: (commands option , principal , 'd', 'e)Form list ->
(commands option , principal , 'd', 'e)Form list]
,'ins:(commands option , principal , 'd', 'e)Form list list ->
'RT_POSITION'].
'outs:commands option list trType list`

414
T R _ e x e c _ c m d _ r u l e

val helper2 = fst(dest_imp(concl helper1))

val RT_POSITION_exec_rtOrient_lemma1 = TAC_PROOF(
  ([], helper2),
  REWRITE_TAC |
  CFGInterpret_def, globalAuth_def, stateAuth_def, getSquadLeaderCOM_def,
  getOmniCOM_def, getSquadLeaderCOMx_def, getOmniCOMx_def, inputList_def,
  extractInput_def, MAP, propCommandList_def, extractPropCommand_def,
  satList_CONS, satList_nil,
  GSYM satList_conj |
  THEN
  REWRITE_TAC |
  NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, 
  output_distinct_clauses, commands_distinct_clauses, 
  principal_distinct_clauses, squadLeaderCom_distinct_clauses, 
  omniCom_distinct_clauses, commands_one_one, 
  GSYM state_distinct_clauses, 
  GSYM output_distinct_clauses, 
  GSYM commands_distinct_clauses, 
  GSYM principal_distinct_clauses, 
  GSYM squadLeaderCom_distinct_clauses, 
  GSYM omniCom_distinct_clauses |
  THEN
  REWRITE_TAC |
  satList_CONS, satList_nil |
  THEN
  PROVE_TAC |
  Controls, Modus_Ponens |
)

val = save_thm("RT_POSITION_exec_rtOrient_lemma1",
  RT_POSITION_exec_rtOrient_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val RT_POSITION_exec_rtOrient_lemma2 = TAC_PROOF(
  ([], helper3),
  PROVE_TAC |
  RT_POSITION_exec_rtOrient_lemma1, 
  TR_exec_cmd_rule |
)

val = save_thm("RT_POSITION_exec_rtOrient_lemma2",
  RT_POSITION_exec_rtOrient_lemma2)

val RT_POSITION_exec_rtOrient_thm = REWRITE_RULE |
  propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def,
  MAP |
  RT_POSITION_exec_rtOrient_lemma2

val = save_thm("RT_POSITION_exec_rtOrient_thm",
  RT_POSITION_exec_rtOrient_thm)

(* State: RT_ORIENTATION *)
(* Execute transition justified: *)
(* RT_ORIENTATION --> RT_ALERT *)

val helper1 = ISPECL |
  (`authentication: (commands option, principal, 'd, 'e)Form -> bool`,
   `globalAuth: (commands option, principal, 'd, 'e)Form list ->
   (commands option, principal, 'd, 'e)Form list`,
```ocaml
val helper2 = fst(dest_imp(concl helper1))

val RT_ORIENTATION_exec_rtAlert_lemma1 = TAC_PROOF(
  ([],helper2),
  REWRITE_TAC[CFGInterpret_def, globalAuth_def, stateAuth_def, getSquadLeaderCOM_def, getOmniCOM_def, getSquadLeaderCOMx_def, getOmniCOMx_def, inputList_def, extractInput_def, MAP, propCommandList_def, extractPropCommand_def, satList_CONS, satList_nil, GSYM satList_conj]
  THEN REWRITE_TAC[NOT_NONE_SOME, NOT_SOME_NONE, SOME_11, state_distinct_clauses, output_distinct_clauses, commands_distinct_clauses, principal_distinct_clauses, squadLeaderCom_distinct_clauses, omniCom_distinct_clauses, commands_one_one, GSYM state_distinct_clauses, GSYM output_distinct_clauses, GSYM commands_distinct_clauses, GSYM principal_distinct_clauses, GSYM squadLeaderCom_distinct_clauses, GSYM omniCom_distinct_clauses]
  THEN REWRITE_TAC[
    satList_CONS, satList_nil]
  THEN PROVE_TAC[Controls, Modus_Ponens]
)

val _ = save_thm("RT_ORIENTATION_exec_rtAlert_lemma1", RT_ORIENTATION_exec_rtAlert_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val RT_ORIENTATION_exec_rtAlert_lemma2 = TAC_PROOF(
  ([],helper3),
  PROVE_TAC[
    RT_ORIENTATION_exec_rtAlert_lemma1, TR_exec_cmd_rule]
)

val _ = save_thm("RT_ORIENTATION_exec_rtAlert_lemma2", RT_ORIENTATION_exec_rtAlert_lemma2)

val RT_ORIENTATION_exec_rtAlert_thm = REWRITE_RULE[
  propCommandList_def, inputList_def, extractPropCommand_def, extractInput_def, MAP]
  RT_ORIENTATION_exec_rtAlert_lemma2
val _ = save_thm("RT_ORIENTATION_exec_rtAlert_thm", RT_ORIENTATION_exec_rtAlert_thm)
```

```
```
val helper1 = ISPECL
  [\'authentication:(commands option ,principal ,d','e)Form -> bool\',
   \'globalAuth:(commands option ,principal ,d','e)Form list ->
     (commands option ,principal ,d','e)Form list\',
   \'stateAuth: state ->
     (commands option ,principal ,d','e)Form list ->
     (commands option ,principal ,d','e)Form list\',
   \'(Name SquadLeader) says (prop (SOME (SquadLeaderCOM complete)))
     :(commands option ,principal ,d','e)Form\',
   \'outs:commands option list trType list\',
   \'RT_ALERT\',
   \'ins:(commands option ,principal ,d','e)Form list list\',
   \'RT_ALERT_exec_complete_lemma1\',
   \'RT_ALERT_exec_complete_lemma2\',
   \'RT_ALERT_exec_complete_thm\']
  TR_exec_cmd_rule

val helper2 = fst(dest_imp(concl helper1))

val RT_ALERT_exec_complete_lemma1 =
TAC_PROOF(
  ([],helper2),
  REWRITE_TAC[CFGInterpret_def ,
    globalAuth_def , stateAuth_def ,
    getSquadLeaderCOM_def ,
    getOmniCOM_def ,
    getSquadLeaderCOMx_def ,
    getOmniCOMx_def ,
    inputList_def ,
    extractInput_def ,
    MAP ,
    propCommandList_def ,
    extractPropCommand_def ,
    satList_CONS ,
    satList_nil , GSYM satList_conj]
  THEN
  REWRITE_TAC[output_distinct_clauses ,
    state_distinct_clauses ,
    commands_distinct_clauses ,
    principal_distinct_clauses ,
    squadLeaderCom_distinct_clauses ,
    omniCom_distinct_clauses ,
    commands_one_one,
    state_distinct_clauses ,
    output_distinct_clauses ,
    commands_distinct_clauses ,
    principal_distinct_clauses ,
    squadLeaderCom_distinct_clauses ,
    omniCom_distinct_clauses]
  THEN
  REWRITE_TAC[satList_CONS ,
    satList_nil] 
  THEN
  PROVE_TAC[
    Controls ,
    Modus_Ponens]
) 

val _ = save_thm("RT_ALERT_exec_complete_lemma1",
                   RT_ALERT_exec_complete_lemma1)

val helper3 = snd(dest_imp(concl helper1))

val RT_ALERT_exec_complete_lemma2 =
TAC_PROOF(
  ([],helper3),
  PROVE_TAC[
    RT_ALERT_exec_complete_lemma1 ,
    TR_exec_cmd_rule]
) 

val _ = save_thm("RT_ALERT_exec_complete_lemma2",
                   RT_ALERT_exec_complete_lemma2)

val RT_ALERT_exec_complete_thm =
REWRITE_RULE[
  propCommandList_def ,
  inputList_def ,
  extractPropCommand_def ,
  extractInput_def ,
  MAP]
  RT_ALERT_exec_complete_lemma2
val _ = save_thm("RT_ALERT_exec_complete_thm", RT_ALERT_exec_complete_thm)

val _ = export_theory();
end
References


https://acronyms.thefreedictionary.com/GOTWA

https://en.wiktionary.org/wiki/METT-TC

https://meltdownattack.com/meltdown.pdf


OBJECTIVES
A career with an organization that values diversity, service, and innovation. Preference for intelligence analysis, offensive & defensive cybersecurity, artificial intelligence & machine learning, research & investigative work, and oversees opportunities in an Arabic-speaking country.

EDUCATION
M.S. Computer Science & M.S. Biomedical Forensic Science May 2019(anticipated) & May 2017
Syracuse University, Syracuse NY
GPA 3.65/4.00 & 3.92/4.00. Master thesis in systems & mission assurance

Post-baccalaureate premedical program August 2010
University of Southern California, Los Angeles CA
GPA 3.42/4.00

Dual B.S. physics & applied computational mathematics June 2006
University of California, Irvine, Irvine CA

CURRENT EXPERIENCES
Master Thesis/Research Assistant in mission assurance December 2016 - present
College of Engineering and Computer Science at Syracuse University, Syracuse NY
Master Thesis in systems/mission assurance Summer 2017 - present
• Assess and analyze stakeholder needs and identify assets and unacceptable asset losses.
• Apply hazard and vulnerability analysis to avoid or mitigate asset losses and design trustworthy CONOPS for missions
• Demonstrate above analysis on a non-automated, human-centered mission (U.S. Army Ranger patrol base operations)
• Develop mission CONOPS into a transitions system and formally verify, using computer-aided theorem prover, that mission CONOPS satisfy security properties with respect to a security policy.
• Verify and document methodology
• Write and defend master thesis

Automate assurance methodology Summer 2018 - October 2018
Assured Information Security, Rome NY
Automate assurance methodology
• Work with small team to automate a mission assurance methodology for transforming a transition system description of a UAV mission into a formally verified hardware description
• Prepare documentation

ADDITIONAL EXPERIENCES
IARPA CREATE Better Reasoning participant August 2018
online: https://www.iarpa.gov/index.php/research-programs/create
• Contributed to research in creating better intelligence analysis tools by testing new tools to analyze evidence, draw conclusions, collaborate, and write a report. Provided feedback to researchers

Independent Study research project in drug delivery techniques Spring 2013
Pasadena City College, Pasadena CA
• Explored medical literature for state-of-the-art technologies in cell-targeting mechanisms (i.e., drug delivery techniques).
• Interested in the use of such technologies as weapons
• Presented to peers orally and to academic advisor in writing

Senior Thesis in plasma fusion physics June 2006
University of California, Irvine, Irvine CA
• Analyzed experimental data from DIII-D tokamak and computer simulations to identify correlations and explain observations.
• Presented to peers, family, and faculty orally, via poster, and in writing. Published 2010

Engineering Aide Fall 2000/Spring 2001
Space Sciences Laboratory, Berkeley CA
• Investigated the effects of chemical modification of microchannel plates (signal amplifiers) to improve quantum efficiency.
• Researched literature and performed data analysis

SKILLS AND LANGUAGES
Recent Programming Experiences Haskell, HOL/ML, Isabelle, C/C++, Python, TensorFlow, Matlab
**Previous Programming Experiences**  Java, Pascal, IDL, Mathematica

**Computer**  MAC OS-X, Windows, Unix/Ubuntu/bash, VirtualBox/VMWare, MS Office, LaTeX, Wireshark, machine learning

**Mathematics**  Calculus (I, II, & III), differential equations, probability & statistics, linear algebra, discrete math, numerical analysis, complex analysis, Fourier analysis, various applied computational mathematics

**Languages**  English (native), Arabic (starting 2nd year at university), Spanish (1-semester in high school and college)

**EXTRACURRICULAR AND OTHER ACTIVITIES**

Syracuse University, Syracuse NY
- Senator-at-large and Diversity Committee Chair of Graduate Student Organization (GSO)
- Serving third year on University Senate Committee on Diversity
- Academic Program Senator (GSO) for biomedical forensic science program

Pasadena City College, Pasadena, CA
-Received special award for volunteering to help students in chemistry

University of Southern California, Los Angeles, CA
- Trojan Heath Volunteer at Los Angeles County Department of The Coroner. Assisted with autopsies

University of California, Irvine, Irvine CA
- President of Society of Physics Students

Golden West College, Huntington Beach, CA
- President and social committee chair of Alpha Gamma Sigma Honor Society

Volunteered at New York State Fair for our local chapter of IEEE

Volunteered to translate Arabic at food bank at University United Methodist Church

Volunteered to care for children of Burmese immigrants while parents studied English at First English Lutheran Church